Abstract

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

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

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

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

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

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

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

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

0.4 Table of Contents

0 Document Information ................................................................. 2
  0.1 License .................................................................................. 2
  0.2 Trademarks ............................................................................ 3
  0.3 Feedback ............................................................................... 3
  0.4 Table of Contents .................................................................... 3
  0.5 Terminology and Document Conventions ...................................... 4
  0.6 Revision History ...................................................................... 4

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

2 Application Domain .................................................................... 5

3 Problem Description .................................................................... 6

4 Requirements ............................................................................. 6

5 Technical Solution ....................................................................... 6

6 Considered Alternatives ............................................................... 12

7 Security Considerations .............................................................. 12

8 Document Support ....................................................................... 12
0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

0.6 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>06/22/12</td>
<td>Initial Draft</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alasdair Nottingham, IBM, <a href="mailto:not@uk.ibm.com">not@uk.ibm.com</a></td>
</tr>
<tr>
<td>0.1</td>
<td>28/09/12</td>
<td>Updates from Basel face to face</td>
</tr>
<tr>
<td>0.2</td>
<td>4th April 2013</td>
<td>Updates for draft publication (largely contract name style)</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>0.3</td>
<td>12th September 2013</td>
<td>Updates from Hursley face to face. Removed javax.servlet.resources and javax.servlet.jsp.resources which only exist in the tomcat version of the API jar.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated the Servlet 3 example.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added JSR numbers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alasdair Nottingham, IBM, <a href="mailto:not@uk.ibm.com">not@uk.ibm.com</a></td>
</tr>
<tr>
<td>0.4</td>
<td>September 16, 2013</td>
<td>Prepare for vote</td>
</tr>
<tr>
<td>1.0</td>
<td>2 October 2013</td>
<td>Final for voting.</td>
</tr>
</tbody>
</table>
1 Introduction

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

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

2 Application Domain

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

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

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

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

4 Requirements

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

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

5 Technical Solution

Rather than define package versions for each individual package in all Java platform packages, a task that will be neither fun, nor will produce consensus. This RFC will propose a set of OSGi contracts for the specifications. These will then be versioned according to the JSR version. A client bundle will then import the packages required with no version and express a dependency on the OSGi contract that defines the packages and the exact specification version required. A provider of the contract then expresses every version of the contract they support.
As shown in figure 1, we have three providers of the JavaServlet contract. The first provider provides JavaServlet 2.5, the second provides JavaServlet 3.0. The Servlet 3.0 specification is specifically written to be backwardly compatible, so a Servlet 3.0 runtime can support Servlet 2.5 applications. As a result, it provides two contracts, one for JavaServlet at 2.5 and one for JavaServlet at 3.0. The third one provides a mythical JavaServlet at version 4 that is NOT compatible with JavaServlet 3.0 and JavaServlet 2.5. In this example, WAB A can run against Servlet Provider 1 and 2, WAB B can run against Servlet provider 2, neither can run against Servlet Provider 3. By inverting the versioning scheme for these contracts, when compared with semantic versions for packages, we remove the risk associated with the undefined mechanism the Java Community Process and the JSR committees use when choosing new versions for their specifications. Vendors are then free to version individual packages in accordance with their existing schemes, thus allowing compatibility between their releases, while being able to support portability across different vendors too. The JCP typically versions specifications based more on marketing considerations rather than based on making a statement about the new specifications compatibility with previous releases of the specifications.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>JSR</th>
<th>Packages</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaEJB</td>
<td>2.1</td>
<td>153</td>
<td>javax.ejb</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>javax.ejb.spi</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Example of using contracts for using Servlet APIs
<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>JSR</th>
<th>Packages</th>
<th>Comments</th>
</tr>
</thead>
</table>
| JavaEJB            | 3       | 220 | javax.ejb
                      javax.ejb.spi                         | Compatible with 2.1                                                                       |
| JavaEJB            | 3.1     | 318 | javax.ejb
                      javax.ejb.embeddable
                      javax.ejb.spi
                      javax.interceptor                      | Compatible with 3, 2.1                                                                    |
| JavaEJB Lite       | 3.1     | 318 | javax.ejb
                      javax.interceptor                      | EJBlite is a subset of the EJB specification. The package content is logically subset. A provider of java.ejb is required to also provide this contract. A provider of java.ejb_lite is not required to provide java.ejb. |
| JavaExpressionLanguage | 2.1    | 245 | javax.el                                      |                                                                                           |
| JavaExpressionLanguage | 2.2    | 245 | javax.el                                      | Compatible with 2.1                                                                       |
| JavaCDI            | 1       | 299 | javax.decorator
                      javax.enterprise.context
                      javax.enterprise.context.spi
                      javax.enterprise.event
                      javax.enterprise.inject
                      javax.enterprise.inject.spi
                      javax.enterprise.util                   |                                                                                           |
| JavaJMS            | 1.1     | 914 | javax.jms                                      |                                                                                           |
| JavaJPA            | 1       | 220 | javax.persistence
                      javax.persistence.spi                   |                                                                                           |
| JavaJPA            | 2       | 317 | javax.persistence
                      javax.persistence.criteria
                      javax.persistence.metamodel
                      javax.persistence.spi                   | Compatible with 1                                                                         |
| JavaJCA            | 1.5     | 112 | javax.resource
                      javax.resource.cci
                      javax.resource.spi
                      javax.resource.spi.endpoint
                      javax.resource.spi.security
                      javax.resource.spi.work                 |                                                                                           |
| JavaJCA            | 1.6     | 322 | javax.resource
                      javax.resource.cci
                      javax.resource.spi
                      javax.resource.spi.endpoint
                      javax.resource.spi.security
                      javax.resource.spi.work                 | Compatible with 1.5                                                                       |
| JavaJASPIC         | 1       | 196 | javax.security.auth.message
                      javax.security.auth.message.callback
                      javax.security.auth.message.config
                      javax.security.auth.message.module     |                                                                                           |
| JavaJACC           | 1.1     | 115 | javax.security.jacc                          |                                                                                           |
| JavaJACC           | 1.4     | 115 | javax.security.jacc                          | Compatible with 1.1                                                                       |
| JavaServlet        | 2.5     | 154 | javax.servlet
                      javax.servlet.http                     |                                                                                           |
<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>JSR</th>
<th>Packages</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaServlet</td>
<td>3</td>
<td>315</td>
<td>javax.servlet&lt;br&gt;javax.servlet.annotation&lt;br&gt;javax.servlet.descriptor&lt;br&gt;javax.servlet.http</td>
<td>Compatible with 2.5</td>
</tr>
<tr>
<td>JavaJSP</td>
<td>2</td>
<td>152</td>
<td>javax.servlet.jsp&lt;br&gt;javax.servlet.jsp.el&lt;br&gt;javax.servlet.jsp.tagext</td>
<td></td>
</tr>
<tr>
<td>JavaJSP</td>
<td>2.1</td>
<td>245</td>
<td>javax.servlet.jsp&lt;br&gt;javax.servlet.jsp.el&lt;br&gt;javax.servlet.jsp.tagext</td>
<td>Compatible with 2</td>
</tr>
<tr>
<td>JavaJSP</td>
<td>2.2</td>
<td>245</td>
<td>javax.servlet.jsp&lt;br&gt;javax.servlet.jsp.el&lt;br&gt;javax.servlet.jsp.resources&lt;br&gt;javax.servlet.jsp.tagext</td>
<td>Compatible with 2.1, 2</td>
</tr>
<tr>
<td>JavaJSTL</td>
<td>1</td>
<td>52</td>
<td>javax.servlet.jsp.jstl.core&lt;br&gt;javax.servlet.jsp.jstl.fmt&lt;br&gt;javax.servlet.jsp.jstl.sql&lt;br&gt;javax.servlet.jsp.jstl.tlv</td>
<td></td>
</tr>
<tr>
<td>JavaJSTL</td>
<td>1.1</td>
<td>52</td>
<td>javax.servlet.jsp.jstl.core&lt;br&gt;javax.servlet.jsp.jstl.fmt&lt;br&gt;javax.servlet.jsp.jstl.sql&lt;br&gt;javax.servlet.jsp.jstl.tlv</td>
<td>Compatible with 1</td>
</tr>
<tr>
<td>JavaJSTL</td>
<td>1.2</td>
<td>52</td>
<td>javax.servlet.jsp.jstl.core&lt;br&gt;javax.servlet.jsp.jstl.fmt&lt;br&gt;javax.servlet.jsp.jstl.sql&lt;br&gt;javax.servlet.jsp.jstl.tlv</td>
<td>Compatible with 1.1</td>
</tr>
<tr>
<td>JavaJTA</td>
<td>1.1</td>
<td>907</td>
<td>javax.transaction&lt;br&gt;javax.transaction.xa</td>
<td></td>
</tr>
<tr>
<td>JavaJTAJRE</td>
<td>1.1</td>
<td>907</td>
<td>javax.transaction&lt;br&gt;javax.transaction.xa</td>
<td>This contains a subset of the package javax.transaction. It only contains 3 exceptions. A provider of java.jta MUST also provide this contract. The OSGi system bundle MUST provide this contract.</td>
</tr>
<tr>
<td>JavaBeanValidation</td>
<td>1</td>
<td>303</td>
<td>javax.validation&lt;br&gt;javax.validation.bootstrap&lt;br&gt;javax.validation.constraints&lt;br&gt;javax.validation.groups&lt;br&gt;javax.validation.metadata&lt;br&gt;javax.validation.spi</td>
<td></td>
</tr>
<tr>
<td>JavaJAXRS</td>
<td>1.1</td>
<td>311</td>
<td>javax.ws.rs&lt;br&gt;javax.ws.rs.core&lt;br&gt;javax.ws.rs.ext</td>
<td></td>
</tr>
<tr>
<td>JavaJAXWS</td>
<td>2.1</td>
<td>224</td>
<td>javax.xml.ws&lt;br&gt;javax.xml.ws.handler&lt;br&gt;javax.xml.ws.handler.soap&lt;br&gt;javax.xml.ws.http&lt;br&gt;javax.xml.ws.soap&lt;br&gt;javax.xml.ws.spi&lt;br&gt;javax.xml.ws.wsaddressing</td>
<td>Compatible with 2.1</td>
</tr>
<tr>
<td>JavaJAXWS</td>
<td>2.2</td>
<td>224</td>
<td>javax.xml.ws</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Version</td>
<td>JSR</td>
<td>Packages</td>
<td>Comments</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>-----</td>
<td>----------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>javax.xml.ws.handler</td>
<td></td>
<td></td>
<td>javax.xml.ws.handler</td>
<td></td>
</tr>
<tr>
<td>javax.xml.ws.handler.soap</td>
<td></td>
<td></td>
<td>javax.xml.ws.handler.soap</td>
<td></td>
</tr>
<tr>
<td>javax.xml.ws.http</td>
<td></td>
<td></td>
<td>javax.xml.ws.http</td>
<td></td>
</tr>
<tr>
<td>javax.xml.ws.soap</td>
<td></td>
<td></td>
<td>javax.xml.ws.soap</td>
<td></td>
</tr>
<tr>
<td>javax.xml.ws.spi</td>
<td></td>
<td></td>
<td>javax.xml.ws.spi</td>
<td></td>
</tr>
<tr>
<td>javax.xml.ws.spi.http</td>
<td></td>
<td></td>
<td>javax.xml.ws.spi.http</td>
<td></td>
</tr>
<tr>
<td>javax.xml.ws.wsaddressing</td>
<td></td>
<td></td>
<td>javax.xml.ws.wsaddressing</td>
<td></td>
</tr>
<tr>
<td>JavaJAXBinding</td>
<td>2.1</td>
<td>222</td>
<td>javax.xml.bind</td>
<td></td>
</tr>
<tr>
<td>JavaJAXBinding</td>
<td>2.2</td>
<td>222</td>
<td>javax.xml.bind</td>
<td>Compatible with 2.1</td>
</tr>
<tr>
<td>JavaAnnotation</td>
<td>1</td>
<td>250</td>
<td>javax.annotation</td>
<td></td>
</tr>
<tr>
<td>JavaAnnotation</td>
<td>1.1</td>
<td>250</td>
<td>javax.annotation</td>
<td>Compatible with 1</td>
</tr>
<tr>
<td>JavaInject</td>
<td>1</td>
<td>330</td>
<td>javax.inject</td>
<td></td>
</tr>
</tbody>
</table>

**Examples**

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

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

**Listing 1: A bundle that uses JavaServlet 3.**

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

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

**Listing 2: A bundle that provides servlet packages unversioned**
Bundle-SymbolicName: jee.vendor2
Bundle-Version: 3
Export-Package: javax.servlet; version=2.6, javax.servlet.http; version=2.6
Provide-Capability: osgi.contract; osgi.contract=JavaServlet; version:Version=3;
    uses:="javax.servlet, javax.servlet.http", osgi.contract;
    osgi.contract=JavaServlet; version:Version=2.5; uses:="javax.servlet, javax.servlet.http"

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

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

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

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

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

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

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

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

Listing 6: A bundle that uses JavaServlet 2.5.

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

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

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

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

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

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

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

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

7  Security Considerations

TBD

8  Document Support

8.1 References

8.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Alasdair Nottingham</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:not@uk.ibm.com">not@uk.ibm.com</a></td>
</tr>
</tbody>
</table>

8.3 Acronyms and Abbreviations

8.4 End of Document
Abstract

This RFC describes a REST interface for managing OSGi framework instances, e.g., in cloud computing setups.
# 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>3 Problem Description</td>
<td>5</td>
</tr>
<tr>
<td>3.1 REST Design</td>
<td>5</td>
</tr>
<tr>
<td>3.2 REST Elements</td>
<td>5</td>
</tr>
<tr>
<td>3.3 The semantics of selected HTTP methods</td>
<td>5</td>
</tr>
<tr>
<td>3.4 REST vs. OSGi</td>
<td>6</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 Resources</td>
<td>7</td>
</tr>
<tr>
<td>5.1.1 The Framework Startlevel Resource</td>
<td>7</td>
</tr>
<tr>
<td>5.1.2 The Bundles Resource</td>
<td>7</td>
</tr>
<tr>
<td>5.1.3 The Bundle Resource</td>
<td>9</td>
</tr>
<tr>
<td>5.1.4 The Bundle State Resource</td>
<td>10</td>
</tr>
<tr>
<td>5.1.5 The Bundle Header Resource</td>
<td>10</td>
</tr>
<tr>
<td>5.1.6 The Bundle Startlevel Resource</td>
<td>11</td>
</tr>
<tr>
<td>5.1.7 The Services Resource</td>
<td>11</td>
</tr>
<tr>
<td>5.1.8 The Service Resource</td>
<td>12</td>
</tr>
<tr>
<td>5.2 Representations</td>
<td>12</td>
</tr>
<tr>
<td>5.2.1 Bundles Representations</td>
<td>13</td>
</tr>
<tr>
<td>5.2.2 The Bundle State Representation</td>
<td>13</td>
</tr>
<tr>
<td>5.2.3 The Bundle Header Representation</td>
<td>14</td>
</tr>
<tr>
<td>5.2.4 The Framework Startlevel Representation</td>
<td>14</td>
</tr>
<tr>
<td>5.2.5 The Bundle Startlevel Representation</td>
<td>15</td>
</tr>
<tr>
<td>5.2.6 The Service Representation</td>
<td>15</td>
</tr>
<tr>
<td>5.2.7 Service Representation</td>
<td>17</td>
</tr>
<tr>
<td>5.2.8 BundleException Representation</td>
<td>17</td>
</tr>
<tr>
<td>5.3 Resource Identifier Overview</td>
<td>18</td>
</tr>
<tr>
<td>5.4 Content Type Matching</td>
<td>18</td>
</tr>
<tr>
<td>5.5 Versioning and Interoperability</td>
<td>19</td>
</tr>
<tr>
<td>5.6 API Extensions</td>
<td>19</td>
</tr>
<tr>
<td>5.6.1 The Extensions Resource</td>
<td>19</td>
</tr>
</tbody>
</table>
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>01/20/12</td>
<td>Initial draft</td>
</tr>
<tr>
<td>2nd draft</td>
<td>09/08/12</td>
<td>Update for the Basel F2F</td>
</tr>
<tr>
<td>3rd draft</td>
<td>09/20/12</td>
<td>Update based on the discussions from the Basel F2F</td>
</tr>
<tr>
<td>4th draft</td>
<td>01/28/13</td>
<td>Update based on the Orlando F2F discussions</td>
</tr>
<tr>
<td>5th draft</td>
<td>02/25/13</td>
<td>Update based on the Austin F2F discussions</td>
</tr>
<tr>
<td>6th draft</td>
<td>06/07/13</td>
<td>Update for the Palo Alto F2F</td>
</tr>
<tr>
<td>7th draft</td>
<td>09/10/13</td>
<td>Update for the Hursley F2F</td>
</tr>
<tr>
<td>8th draft</td>
<td>01/16/14</td>
<td>Update for the Austin F2F</td>
</tr>
</tbody>
</table>
1 Introduction

Cloud computing is a continuing trend in the IT industry. As discussed in RFP 133 [3], OSGi appears to be an ideal base for building scalable and dependable applications for the cloud. One of the possible scenarios for OSGi to be successfully applied to cloud computing is to use it in a Platform as a Service spirit. Users write their bundles and can deploy them to their own OSGi instance running in the cloud. This, however, requires the platform provider to expose the OSGi management API to the end user and make them available through a network protocol.

One of the popular approaches in cloud computing to remote communication is the use of RESTful web services. This document discusses the technical background of REST, the applicability to managing OSGi frameworks through a REST-style API, and proposes a concrete protocol.

2 Application Domain

Cloud Computing is a major IT trend; possibly qualifying as a real paradigm shift whose eventual impact may exceed that of the move to client server computing in the 1980's. Cloud Computing service are already extensively used to provide cost effective third party hosting for some categories of traditional application; differing only in the granularity of utilization and subsequent payment from the previous utility compute service providers. However, for applications architected in the appropriate manner, Cloud Computing also offers the promise of massive 'just-in-time' resource elasticity.

The JCP is preparing for Cloud support in future versions of JavaSE and JavaEE. JSR 342 (http://jcp.org/en/jsr/detail?id=342), the JavaEE 7 JSR, makes cloud its main theme, mentioning PaaS support, multi-tenancy and elasticity. To properly support multitenancy within the VM some form of modularity and isolation is required and OSGi modularity is available for JavaSE today.

Managing OSGi frameworks has been an important problem for traditional deployments on desktop machines, mobile devices or servers. The one side of the problem is monitoring a running OSGi framework instance, e.g., to ensure correctness of a deployment. In public cloud environments, this capability is essential since they are typically metered in some form and unwanted behavior can lead to monetary losses. The other side of framework management is the deployment. Being able to bootstrap the framework with a set of bundles has been shown to be important and lead to the inclusion of the launch API. In a PaaS setup, the management interface can be the only way of interacting with the framework instance and a prerequisite for deploying application modules.

Given the particular importance of management for cloud setups, it is clear that some form is needed and since the management interface is client-facing it needs to be standardized for interoperability between...
OSGi cloud offerings. Furthermore, existing protocols do not work well with wide-area networks and their firewall restrictions. Therefore, this RFC discusses the use of REST, a protocol akin to HTTP and widely used in interaction with clouds.

## 3 Problem Description

Representational State Transfer (REST) [4] is the architectural style of the world wide web. It can be described as a set of constraints that govern the interactions between the main components of the Internet. Recently, REST style interaction has gained popularity as an architecture for web services (RESTful web services), mainly to overcome the perceived complexity and verbosity of SOAP-based web services [5].

### 3.1 REST Design

**Client-Server** is a separation of concern between the entity responsible for the user-interaction (client) and the other entity (server) responsible for data storage. For instance, in the original world wide web the browser is the client rendering and presenting the content delivered by one or more web servers. As a result, web content becomes more portable and content providers more scalable.

**Stateless** State is entirely kept at the client side. Therefore, every request must contain all state required for the server to accomplish the transaction and deliver content. The main rationale behind this design constraint is to again improve the scalability since in a pure stateless design the server resources are not burdened with maintaining any client state. Another perceived advantage is that the failure models of stateless interactions is simpler and fault tolerance easier to achieve.

**Cacheable** Content marked as cacheable can be temporarily stored and used to immediately answer future equivalent requests and improve efficiency and reduce network utilization and access latencies. Due to the end-to-end principle, caches can be placed where necessary, e.g., at the client (forward-proxy), or at the server side (backward-proxy).

**Layered** Layering introduces natural boundaries to coupling since every layer only accesses the services provided by the lower layer and provides services to the next higher layer.

**Uniform Interface** Generality of component interfaces provides a natural decoupling of implementation and interface. REST furthermore encourages the separation of identifiable resources (addressing) and their representation (content delivery).

### 3.2 REST Elements

**Resources and Resource Identifiers:** A resource is an abstract piece of information that can be addressed
by a resource identifier. The mapping of a resource to a concrete set of entities can vary over time.

**Representation:** A representation is a sequence of bytes plus associated meta-data that describe the state of a resource. The data format of a representation is called the *media-type*. Every concrete representation of a resource is just one of arbitrarily many possible representations. The selection of a concrete representation of a resource can be made according to the media types supported by both the client and the server.

### 3.3 The semantics of selected HTTP methods

**GET** retrieves the representation addressed by the request URI. The operation is expected to be idempotent.

**POST** add the enclosed entity as a new subordinate of the resource identified by the request URI.

**PUT** store the enclosed entity under the request URI. The operation is expected to be idempotent.

**DELETE** delete the resource identified by the request URI.

### 3.4 REST vs. OSGi

Scalability is unlikely to become a significant issue for an OSGi framework as a REST server. The number of people that concurrently want to manage the same OSGi framework is not expected to be high.

What is an issue though is portability.

Users likely want to use a diversity of programming and scripting languages or even embed the management functionality into web pages.

Furthermore, the use of an open-standard protocol like HTTP that is able to easily penetrate firewalls as well as the frequent use of open data formats like XML or JSON as a base for the media types makes REST-style APIs appealing to users.

In this regard, a REST-style API can help to support the adoption of OSGi in the cloud. Arguably, cloud users are used to some flavor of REST-style APIs so that not having one could in turn hinder acceptance of OSGi in the cloud.

One can, however, generally question the wisdom in treating an OSGi framework as a set of resources. REST is centered around a data model that is hypertext-driven and many existing HTTP-based APIs that call themselves RESTful have been criticized for not complying with this principle [6].

As a consequence, some operations that are straightforward in an object model like the traditional OSGi APIs (e.g., starting a bundle) require some re-thinking when mapping them to a resource-centric API.

Starting a bundle now becomes an update to the state resource of a particular bundle resource.

One can particularly question if there is any way at all to make an OSGi framework present itself in a way that does not create coupling between client and server, one of the important goals of REST. After all, the proposed API is about managing OSGi frameworks and that in fact requires knowledge about OSGi and the operation it supports.

The pessimistic conclusion might be that it is neither wise nor fully possible to use pure REST for the management of an OSGi framework.

So, in other words: "Whereof one cannot speak therof one must be silent" [7].?

This proposal takes a pragmatic approach: REST-style APIs (may they be compliant or not) are
successfully used for similar purposes. This proposal is intended to be a starting point. A discussion about the compliance with the REST design has been initiated.

4 Requirements

- MAN0001: The solution SHOULD define APIs to interact with a variety of different cloud platforms to manage the resource pool by, e.g., adding new nodes or removing nodes, where nodes represent OSGi Frameworks.
- MAN0002: The solution SHOULD define APIs that allow querying of deployed Systems (Composite Applications), System Elements (Subsystems), Bundles and Services on an OSGi Framework in the Cloud.
- MAN0003: The deployment of bundles SHOULD facilitate consistent behavior across multiple framework instances. It SHOULD be possible to receive feedback on the deployment status.
- MAN0004: – The solution MUST provide APIs to discover available OSGi Frameworks in a Resource Domain.

5 Technical Solution

The following API embraces three important management tasks:

- **introspecting** a running OSGi framework, the bundles installed, and the services available.
- **changing the state** of the framework by installing new bundles or changing the state of an existing bundle
- **dealing with the startlevel** of the framework and the target startlevels of bundles.

5.1 Resources

The framework and its state is mapped to a set of different resources. Each resource is accessible through a resource identifier, as summarized in 5.3.
5.1.1 The Framework Startlevel Resource
The startlevel resource represents the active start level of the framework. It supports the following requests:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GET]</td>
<td>get the startlevel. Returns a <code>bundleframework</code> startlevel representation (5.2.5).</td>
</tr>
</tbody>
</table>

**Status codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
<tr>
<td>406 (NOT ACCEPTABLE)</td>
<td>does not support any of the requested representations</td>
</tr>
</tbody>
</table>

| [PUT]  | set the startlevel. Expects the body of the message to be a `bundleframework` startlevel representation (5.2.5). |

**Status codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204 (NO CONTENT)</td>
<td>success</td>
</tr>
<tr>
<td>415 (UN SUPPORTED MEDIA TYPE)</td>
<td>the request has a media type that is not supported</td>
</tr>
<tr>
<td>500 (INTERNAL SERVER ERROR)</td>
<td>the operation has thrown an exception, e.g., IllegalArgumentException when the requested start level is less or equal to zero.</td>
</tr>
</tbody>
</table>

5.1.2 The Bundles Resource
The bundles resource represents the set of all bundles installed on the framework. The resource supports the following requests:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GET]</td>
<td>request the list of installed bundles in the form of the bundle list representation (5.2.1).</td>
</tr>
</tbody>
</table>

**Status codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
<tr>
<td>406 (NOT ACCEPTABLE)</td>
<td>does not support any of the requested representations</td>
</tr>
</tbody>
</table>

| [POST] | install a bundle addressed by a `URL-location string`. The body of the message is expected to be a plain `URL` string with mime type text/plain. The `URL` and `location` will be used as the location of the bundle. In practice, this is often a `URL` and must be reachable by the target. Returns the (local) `URL` of the newly installed bundle. |

**Status codes:**
### Status codes:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
<tr>
<td>400 (BAD REQUEST)</td>
<td>if the request body is not a valid OSGi bundle</td>
</tr>
<tr>
<td>409 (CONFLICT)</td>
<td>if there is already a bundle installed with the same location string</td>
</tr>
<tr>
<td>500 (INTERNAL SERVER ERROR)</td>
<td>the operation has thrown a bundle exception. The body of the message MUST be a BundleException representation 5.2.8.</td>
</tr>
</tbody>
</table>

### Method:

#### POST

Install the bundle that is uploaded as the body of this request. Media-type SHOULD be vnd.osgi.bundle. Implementations are free to accept additional mime types other than text/plain such as application/zip or application/x-jar. Implementations MUST generate a random location string that is passed to the installBundle (core 10.1.8.18) method in order to avoid unintended collisions between unrelated bundles. Returns the (local) URI of the newly installed bundle.

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
<tr>
<td>400 (BAD REQUEST)</td>
<td>if the request body is not a valid OSGi bundle</td>
</tr>
<tr>
<td>409 (CONFLICT)</td>
<td>if there is already a bundle installed with the same location string</td>
</tr>
<tr>
<td>500 (INTERNAL SERVER ERROR)</td>
<td>the operation has thrown a bundle exception. The body of the message MUST be a BundleException representation 5.2.8.</td>
</tr>
</tbody>
</table>

The bundles resource returns a list of the URIs of all bundles installed on the framework. For clients interested in all bundles there is also the possibility to retrieve the bundle representation of each installed bundle with a single request through the bundles/representations resource:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>request the representations of all installed bundles in the form of a bundle representations list (5.2.1.2).</td>
</tr>
</tbody>
</table>

#### Status codes:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
<tr>
<td>406 (NOT ACCEPTABLE)</td>
<td>does not support any of the requested</td>
</tr>
</tbody>
</table>
### 5.1.3 The Bundle Resource

The bundle resource represents a single, distinct bundle in the system. Hence, it has to be qualified by a bundle id. The bundle resource supports the following requests:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GET]</td>
<td>request the bundle representation (5.2).</td>
</tr>
</tbody>
</table>

**Status codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
<tr>
<td>404 (NOT FOUND)</td>
<td>there is no bundle with this bundle id</td>
</tr>
<tr>
<td>406 (NOT ACCEPTABLE)</td>
<td>does not support any of the requested representations</td>
</tr>
</tbody>
</table>

| [PUT] | update the bundle with a new version. The body of the message is expected to be a plain URL with mime type text/plain referencing the bundle that is to be used for updating the existing bundle or an empty content to trigger bundle.update(). |

**Status codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204 (NO CONTENT)</td>
<td>success</td>
</tr>
<tr>
<td>400 (BAD REQUEST)</td>
<td>URL could not be parsed or the bundle could not be retrieved from the given URL if the URL is invalid</td>
</tr>
<tr>
<td>404 (NOT FOUND)</td>
<td>there is no bundle with this bundle id</td>
</tr>
<tr>
<td>500 (INTERNAL SERVER ERROR)</td>
<td>the operation has thrown a bundle exception. The body of the message MUST be a BundleException representation 5.2.8.</td>
</tr>
</tbody>
</table>

| [PUT] | updates the content of the bundle with the content in the body of this request. Media-type SHOULD be vnd.osgi.bundle. Implementations are free to accept additional mime types other than text/plain such as application/zip or application/x-jar. |

**Status codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
<tr>
<td>400 (BAD REQUEST)</td>
<td>if the request body is not a valid OSGi bundle</td>
</tr>
<tr>
<td>500 (INTERNAL SERVER ERROR)</td>
<td>the operation has thrown a bundle exception. The body of the message MUST be a BundleException representation 5.2.8.</td>
</tr>
</tbody>
</table>
install/uninstall the bundle.

<table>
<thead>
<tr>
<th>Status codes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>204 (NO CONTENT)</td>
</tr>
<tr>
<td>404 (NOT FOUND)</td>
</tr>
<tr>
<td>500 (INTERNAL SERVER ERROR)</td>
</tr>
</tbody>
</table>

5.1.4 The Bundle State Resource

The bundle state resource represents the state of an installed bundle qualified through its bundle id. It supports the following requests:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>request the state of the bundle as a bundle status representation (5.2.2).</td>
</tr>
<tr>
<td>PUT</td>
<td>attempts to start the bundle (setting the state to 32=started) or stop the bundle (setting the state to 4=resolved). Returns the actual new state in bundle state representation (5.2.2).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status codes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
</tr>
<tr>
<td>404 (NOT FOUND)</td>
</tr>
<tr>
<td>406 (NOT ACCEPTABLE)</td>
</tr>
<tr>
<td>400 (BAD REQUEST)</td>
</tr>
<tr>
<td>404 (NOT FOUND)</td>
</tr>
<tr>
<td>406 (NOT ACCEPTABLE)</td>
</tr>
<tr>
<td>415 (UNSUPPORTED MEDIA TYPE)</td>
</tr>
<tr>
<td>500 (INTERNAL SERVER ERROR)</td>
</tr>
</tbody>
</table>

5.1.5 The Bundle Header Resource

The bundle header resource represents the manifest header of a bundle. The bundle header resource supports the following requests:
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GET]</td>
<td>get the bundle header representation (5.2.3).</td>
</tr>
</tbody>
</table>

**Status codes:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
<tr>
<td>404 (NOT FOUND)</td>
<td>there is no bundle with this bundle id</td>
</tr>
<tr>
<td>406 (NOT ACCEPTABLE)</td>
<td>does not support any of the requested representations</td>
</tr>
</tbody>
</table>

### 5.1.6 The Bundle Startlevel Resource

The startlevel resource represents the start level of a specific bundle. It supports the following requests:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GET]</td>
<td>get the startlevel. Returns a startlevel representation (5.2.5).</td>
</tr>
</tbody>
</table>

**Status codes:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
<tr>
<td>404 (NOT FOUND)</td>
<td>there is no bundle with this bundle id</td>
</tr>
<tr>
<td>406 (NOT ACCEPTABLE)</td>
<td>does not support any of the requested representations</td>
</tr>
</tbody>
</table>

| [PUT] | set the startlevel. Expects the body of the message to be a startlevel representation (5.2.5). |

**Status codes:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204 (NO CONTENT)</td>
<td>success</td>
</tr>
<tr>
<td>404 (NOT FOUND)</td>
<td>there is no bundle with this bundle id</td>
</tr>
<tr>
<td>415 (UNSUPPORTED MEDIA TYPE)</td>
<td>the request has a media type that is not supported</td>
</tr>
<tr>
<td>500 (INTERNAL SERVER ERROR)</td>
<td>the operation has thrown an exception, e.g., IllegalArgumentException when the requested start level is less or equal to zero or the bundle is the system bundle. If the caught exception was a BundleException, a BundleException representation 5.2.8. MUST be returned.</td>
</tr>
</tbody>
</table>

Implementations MUST ignore the values for persistentlystarted and activationpolicyused in PUT request bodies.
5.1.7 The Services Resource
The services resource represents the set of all services available on the framework, optionally constrained by those matching a given filter expression. The resource supports the following requests:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GET]</td>
<td>request the list of available services in the form of one of the service list representation (5.2.7.1).</td>
</tr>
</tbody>
</table>

**Status codes:**

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
<tr>
<td>400 (BAD REQUEST)</td>
<td>the filter expression was not valid (only if a filter is used)</td>
</tr>
<tr>
<td>406 (NOT ACCEPTABLE)</td>
<td>does not support any of the requested representations</td>
</tr>
</tbody>
</table>

The services resource returns a list of the URIs of all services available on the framework. For clients interested in all services there is also the possibility to retrieve the service representation of each available service with a single request through the services/representations resource:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GET]</td>
<td>request the representations of all available services in the form of a service representations list (5.2.7.2).</td>
</tr>
</tbody>
</table>

**Status codes:**

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
<tr>
<td>400 (BAD REQUEST)</td>
<td>the filter expression was not valid (only if a filter is used)</td>
</tr>
<tr>
<td>406 (NOT ACCEPTABLE)</td>
<td>does not support any of the requested representations</td>
</tr>
</tbody>
</table>

5.1.8 The Service Resource
The service resource represents a single, distinct service in the system. Hence, it has to be qualified by a service id. The service resource supports the following requests:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GET]</td>
<td>request the service representation (5.2.6).</td>
</tr>
</tbody>
</table>

**Status codes:**

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
</tbody>
</table>
5.2 Representations

The Bundle Representation

### JSON

Content-Type: `application/org.osgi.bundle+json`

```json
{
  "id":0,
  "lastModified":1314999275542,
  "location":"System Bundle",
  "state":32,
  "symbolicName":"org.eclipse.osgi",
  "version":"3.7.0.v20110613"
}
```

### XML

Content-Type: `application/org.osgi.bundle+xml`

```xml
<bundle>
  <id>0</id>
  <lastModified>1314999275542</lastModified>
  <location>System Bundle</location>
  <state>32</state>
  <symbolicName>org.eclipse.osgi</symbolicName>
  <version>3.7.0.v20110613</version>
</bundle>
```

5.2.1 Bundles Representations

5.2.1.1 The Bundle List Representation

### JSON

Content-Type: `application/org.osgi.bundles+json`

```json
{
  [bundleURI, bundleURI, ..., bundleURI]
}
```

### XML

Content-Type: `application/org.osgi.bundles+xml`

```xml
<bundles>
  <bundleURI/>
  <bundleURI/>
  ...
</bundles>
```
5.2.1.2 The Bundle Representations List Representation

<table>
<thead>
<tr>
<th>JSON</th>
<th>Content-Type: <code>application/org.osgi.bundles.representations+json</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>[BUNDLE REPRESENTATION, BUNDLE REPRESENTATION, ..., BUNDLE REPRESENTATION]</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XML</th>
<th>Content-Type: <code>application/org.osgi.bundles.representations+xml</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;bundles&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>BUNDLE REPRESENTATION</code></td>
</tr>
<tr>
<td></td>
<td><code>BUNDLE REPRESENTATION</code></td>
</tr>
<tr>
<td></td>
<td><code>...</code></td>
</tr>
<tr>
<td></td>
<td><code>BUNDLE REPRESENTATION</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/bundles&gt;</code></td>
</tr>
</tbody>
</table>

5.2.2 The Bundle State Representation

<table>
<thead>
<tr>
<th>JSON</th>
<th>Content-Type: <code>application/org.osgi.bundlestate+json</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>`{</td>
</tr>
<tr>
<td></td>
<td>&quot;state&quot;:32</td>
</tr>
<tr>
<td></td>
<td>&quot;options&quot;:1</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XML</th>
<th>Content-Type: <code>application/org.osgi.bundlestate+xml</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;bundleState&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;state&gt;32&lt;/state&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;options&gt;1&lt;/options&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/bundleState&gt;</code></td>
</tr>
</tbody>
</table>

5.2.3 The Bundle Header Representation
5.2.4 The Framework Startlevel Representation

<table>
<thead>
<tr>
<th>JSON</th>
<th>Content-Type: <code>application/org.osgi.frameworkstartlevel+json</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>`{</td>
</tr>
<tr>
<td></td>
<td>&quot;startLevel&quot;:6,</td>
</tr>
<tr>
<td></td>
<td>&quot;initialBundleStartLevel&quot;:4</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XML</th>
<th>Content-Type: <code>application/org.osgi.frameworkstartlevel+xml</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;frameworkStartLevel&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;startLevel&gt;6&lt;/startLevel&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;initialBundleStartLevel&gt;4&lt;/initialBundleStartLevel&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/frameworkStartLevel&gt;</code></td>
</tr>
</tbody>
</table>

5.2.5 The Bundle Startlevel Representation

<table>
<thead>
<tr>
<th>JSON</th>
<th>Content-Type: <code>application/org.osgi.bundlestartlevel+json</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>`{</td>
</tr>
<tr>
<td></td>
<td>&quot;startLevel&quot;:6</td>
</tr>
<tr>
<td></td>
<td>&quot;activationPolicyUsed&quot;:true</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XML</th>
<th>Content-Type: <code>application/org.osgi.bundlestartlevel+xml</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;bundleStartLevel&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;startLevel&gt;6&lt;/startLevel&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;activationPolicyUsed&gt;true&lt;/activationPolicyUsed&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/bundleStartLevel&gt;</code></td>
</tr>
<tr>
<td><strong>XML</strong></td>
<td>Content-Type: <code>application/org.osgi.bundlestartlevel+xml</code></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><code>&lt;bundleStartLevel&gt;</code>&lt;br&gt;  <code>&lt;startLevel&gt;6&lt;/startLevel&gt;</code>&lt;br&gt;  <code>&lt;activationPolicyUsed&gt;true&lt;/activationPolicyUsed&gt;</code>&lt;br&gt;  <code>&lt;persistentlyStarted&gt;false&lt;/persistentlyStarted&gt;</code>&lt;br&gt;  <code>&lt;/bundleStartLevel&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

5.2.6 The Service Representation

<table>
<thead>
<tr>
<th><strong>JSON</strong></th>
<th>Content-Type: <code>application/org.osgi.service+json</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>`{&lt;br&gt;  properties:&lt;br&gt;  {&lt;br&gt;    key:value,&lt;br&gt;    key:value,&lt;br&gt;    ...&lt;br&gt;    key:value&lt;br&gt;  },&lt;br&gt;  &quot;bundle&quot;:bundleURI,&lt;br&gt;  &quot;usingBundles&quot;:[bundleURI, bundleURI, … bundleURI],&lt;br&gt;  }</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>XML</strong></th>
<th>Content-Type: <code>application/org.osgi.service+xml</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;service&gt;</code>&lt;br&gt;  <code>&lt;properties&gt;</code>&lt;br&gt;    <code>&lt;entry key=&quot;key&quot; value=&quot;value&quot;/&gt;</code>&lt;br&gt;    <code>&lt;entry key=&quot;key&quot; value=&quot;value&quot;/&gt;</code>&lt;br&gt;    ...&lt;br&gt;    <code>&lt;entry key=&quot;key&quot; value=&quot;value&quot;/&gt;</code>&lt;br&gt;  &lt;/properties&gt;<code>&lt;br&gt;  </code>&lt;bundle&gt;bundleURI&lt;/bundle&gt;<code>&lt;br&gt;  </code>&lt;usingBundles&gt;<code>&lt;br&gt;    </code>&lt;bundle&gt;bundleURI&lt;/bundle&gt;<code>&lt;br&gt;    </code>&lt;bundle&gt;bundleURI&lt;/bundle&gt;<code>&lt;br&gt;    ...&lt;br&gt;    </code>&lt;bundle&gt;bundleURI&lt;/bundle&gt;<code>&lt;br&gt;  &lt;/usingBundles&gt;</code>&lt;br&gt;  <code>&lt;/service&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>
5.2.7 Service Representation

5.2.7.1 The Service List Representation

<table>
<thead>
<tr>
<th>Format</th>
<th>Content-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON</td>
<td><code>application/org.osgi.services+json</code></td>
</tr>
<tr>
<td></td>
<td><code>{ [serviceURI, serviceURI, ..., serviceURI] }</code></td>
</tr>
<tr>
<td>XML</td>
<td><code>application/org.osgi.services+xml</code></td>
</tr>
</tbody>
</table>
|        | `<services>
|        |   <uri>serviceURI</uri>
|        |   <uri>serviceURI</uri>
|        |   ...
|        |   <uri>serviceURI</uri>
|        | </services>` |

5.2.7.2 The Service Representations List Representation

<table>
<thead>
<tr>
<th>Format</th>
<th>Content-Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON</td>
<td><code>application/org.osgi.services.representations+json</code></td>
<td><code>[SERVICE REPRESENTATION, SERVICE REPRESENTATION, ..., SERVICE REPRESENTATION]</code></td>
</tr>
</tbody>
</table>
| XML    | `application/org.osgi.services.representations+xml` | `<services>
|        |   SERVICE REPRESENTATION
|        |   SERVICE REPRESENTATION
|        |   ...
|        |   SERVICE REPRESENTATION
|        | </services>` |

5.2.8 BundleException Representation

If the implementation catches a BundleException while retrieving the requested resource, a BundleException representation is returned. The message can be an arbitrary string and is intended to be interpreted by humans. The type code corresponds to the codes defined in
10.1.10 of the core specification and MUST be set by the implementation to the value of the getType() method (10.1.20) on the caught BundleException.

<table>
<thead>
<tr>
<th>JSON</th>
<th>Content-Type: application/org.osgi.bundleexception+json</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;typecode&quot;: 5,</td>
</tr>
<tr>
<td></td>
<td>&quot;message&quot;: &quot;BundleException: Bundle activation error&quot;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XML</th>
<th>Content-Type: application/org.osgi.bundleexception+xml</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;bundleexception&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;typecode&gt;5&lt;/typecode&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;message&gt;BundleException: Bundle activation error&lt;/message&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/bundleexception&gt;</td>
</tr>
</tbody>
</table>

### 5.3 Resource Identifier Overview

framework
framework/state
framework/startlevel
framework/bundles
framework/bundles/representations
framework/bundle/{bundleid}
framework/bundle/{bundleid}/state
framework/bundle/{bundleid}/startlevel
framework/bundle/{bundleid}/header
framework/services
framework/services/representations
framework/service/{serviceid}

framework/bundle/0/state is an alias for framework/state

The bundles, bundles/representations, services, and services/representations resources allow the use of a query parameter which specifies a filter to restrict the result set.
The filter expression follows the IETF RFC 1960 String Representation of LDAP Search Filter format [8].

Filters on services are matched against the service attributes. The query parameter is of the form:

\[ \text{framework/services?filter=ldap-filter} \]

Filters on bundles are matched against the attributes of capabilities in the respective namespaces. Filters on bundles have the form:

\[ \text{framework/bundles?namespace1=ldap-filter1&namespace2=ldap-filter2&...} \]

A missing namespace declaration implies the IdentityNamespace (“osgi.identity”).

### 5.4 Content Type Matching

The solution offering different variants of representations (e.g., JSON or XML) MUST return the best matching variant based on the HTTP accept header. In addition, it MUST respect the file extensions defined for the different media types as specified in the respective IETF RFC (e.g., “.xml” as specified in IETF RFC 3032 and “.json” as specified in IETF RFC 4627). If a file extension is appended to the resource, the solution MUST return the variant mandated by the file extension if it supports this content type.

### 5.5 Versioning and Interoperability

All representations described in this document have version 1. Future versions of the representations MUST contain the version number in their content type (e.g., application/org.osgi.bundlesV2) to allow legacy clients to explicitly request an older version of the representation that they understand by setting their accept header.

Clients MUST understand all attributes described in this documents and MAY ignore any further attribute that the implementation of the REST interface might add.

### 5.6 API Extensions

Implementations MUST allow providers of REST interfaces for components other than the framework to extend the client-facing API. Since there are various possible technologies that can be used for implementing the REST API, e.g., Java Servlet, Restlet, JAX-RS, etc., the extension mechanism can only provide visibility of extensions to the client since any tighter integration of extensions would expose implementation details and therefore limit the choice of technologies that can be used for an implementation.
In order to be discoverable by the client, an extension MUST register itself as a org.osgi.rest.RestApiExtension service through a whiteboard pattern. It MUST set the org.osgi.rest.RestApiExtension.URI_PATH property to the URI under which it can be reached. If the URI is relative, it is interpreted as relative to the “extensions/” path. This URI can either be relative (on the same host and port, tightly integrated with the REST management service) or absolute (separate host and port, independent of the REST management service). The path in the URI must always be absolute.

Extensions providing a REST API for OSGi specifications MUST set org.osgi.rest.RestApiExtension.NAME to the package name of the specification for which they provide the API. All other extensions MUST set the field to a package name but MAY choose a name other than names with “org.osgi*” as a prefix.

Whiteboard pattern registrations under the path “framework” or “extensions” MUST be ignored. Implementations MUST provide the Extensions Resource under the path “extensions”, at the same hierarchical level as the framework resource.

### 5.6.1 The Extensions Resource

The extensions resource enumerates all API extensions that are currently registered through the whiteboard pattern:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GET]</td>
<td>request the list of registered extensions in the form of one of the extension list representation (5.6.2).</td>
</tr>
</tbody>
</table>

**Status codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>success</td>
</tr>
</tbody>
</table>

### 5.6.2 The Extension List Representation

<table>
<thead>
<tr>
<th>Format</th>
<th>Content-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON</td>
<td>application/org.osgi.extensions+json</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>[ { &quot;name&quot; : &quot;org.osgi.service.event&quot;, &quot;path&quot; : &quot;extensions/eventadmin&quot; }, ... ]</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>XML</td>
<td>application/org.osgi.extensions+xml</td>
</tr>
<tr>
<td></td>
<td>&lt;extensions&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;extension&gt;</td>
</tr>
</tbody>
</table>

Copyright © IBM Corporation 2014    All Rights Reserved
6 Considered Alternatives

The only technology that currently exist under the OSGi umbrella and that is designed for performing the remote management of a running OSGi framework is JMX. Arguably, JMX could be used in the cloud since it generally does not prohibit the use of web service protocols but it also does not make particularly easy to use them. The main argument against JMX has to be that REST-style APIs are easier to deal with from a programmer's perspective and are much more adopted in the overall cloud ecosystem.

Another alternative to avoid creating new APIs would be to canonically map the existing JMX APIs to REST, e.g., by implementing a JMX-to-REST bridge. This, however, would suffer from a great conceptual mismatch since JMX is an object-centric architecture whereas REST is resource-centric. The resulting REST-style API would possibly violate each and every REST design principle.

Finally, RFC 140 (Residential Management Tree) from the Residential Expert Group (REG) deals with a similar problem but is targeted to the object-centric TR-069 protocol. There is some overlap of the conceptual model with the proposed REST API but RFC 140 requires a fine-granular micro-management approach while this RFC is focused on the management of OSGi frameworks in the large.

Some particular design decisions were discussed and alternative approaches explored.

First of all, there is duplication in the API with regard to the bundle state. This property of the bundle can either be retrieved through the bundle representation or through the bundle state resource. The reason for this design is that the bundle state is the only mutable property of the bundle. Therefore, it has been modeled as a separate resource that accepts updates through post.

As an alternative, one could make the entire bundle representation mutable and accepting post requests and every update other than on the state property would either be silently ignored or result in an error code to be returned.

Second, there has been discussions about modeling the lifecycle operations on bundles not through a resource at all but through new HTTP methods on the bundle resource. The IETF RFC 2068 explicitly allows other methods (extension methods) for HTTP requests. However, concerns were raised with regard to problems with firewalls or other security mechanisms in the communication path that could
reject any HTTP method that is not explicitly defined in the IETF RFC 2068.

In this context, it was even discussed to abandon the use of PUT and DELETE since some members reported customer to have firewall restrictions in place that only allow GET and POST requests. However, after doing a survey of existing cloud APIs, the result was that all of the APIs of major cloud players (we looked at Amazon {S3, CloudFront, Route 53, and Simple DB}, Rackspace, Microsoft Azure {BlobService, QueueService, and Table Service}; Google App Engine appengine-rest-server; VMWare vCloud) except for the Amazon SimpleDB API make use of PUT and DELETE, some also use HEAD in their API.
7 Javadoc
## Interface RestApiExtension

```
org.osgi.service.rest
public interface RestApiExtension
```

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Error: Reference source not found</td>
</tr>
<tr>
<td>URI_PATH</td>
<td>Error: Reference source not found</td>
</tr>
</tbody>
</table>

### Field Detail

**URI_PATH**

```java
public static final String URI_PATH = "org.osgi.rest.uri.path"
```

**NAME**

```java
public static final String NAME = "org.osgi.rest.name"
```

---

# 8 REST Client Javadoc

Copyright © IBM Corporation 2014 All Rights Reserved
interface RestClient
org.osgi.service.rest.client

All Known Implementing Classes:
RestClientImpl

public interface RestClient

Method Detail

getFrameworkStartLevel

org.osgi.dto.framework.startlevel.FrameworkStartLevelDTO getFrameworkStartLevel()
throws Exception

setFrameworkStartLevel

void setFrameworkStartLevel(org.osgi.dto.framework.startlevel.FrameworkStartLevelDTO startLevel)
throws Exception

getBundles

Collection<String> getBundles()
throws Exception

getBundleRepresentations

Collection<org.osgi.dto.framework.BundleDTO> getBundleRepresentations()
throws Exception

getBundle

org.osgi.dto.framework.BundleDTO getBundle(long id)
throws Exception

getBundle

org.osgi.dto.framework.BundleDTO getBundle(String bundlePath)
throws Exception
getBundleState

int getBundleState(long id)
    throws Exception

    Throws:
    Exception

getBundleState

int getBundleState(String bundlePath)
    throws Exception

    Throws:
    Exception

startBundle

void startBundle(long id)
    throws Exception

    Throws:
    Exception

startBundle

void startBundle(String bundlePath)
    throws Exception

    Throws:
    Exception

startBundle

void startBundle(long id, int options)
    throws Exception

    Throws:
    Exception

startBundle

void startBundle(String bundlePath, int options)
    throws Exception

    Throws:
    Exception

stopBundle

void stopBundle(long id)
    throws Exception

    Throws:
    Exception
stopBundle

void stopBundle(String bundlePath)
throws Exception

Throws:
Exception

stopBundle

void stopBundle(long id, int options)
throws Exception

Throws:
Exception

stopBundle

void stopBundle(String bundlePath, int options)
throws Exception

Throws:
Exception

getBundleHeaders

java.util.Map<String, Object> getBundleHeaders(long id)
throws Exception

Throws:
Exception

getBundleHeaders

java.util.Map<String, Object> getBundleHeaders(String bundlePath)
throws Exception

Throws:
Exception

getBundleStartLevel

org.osgi.dto.framework.startlevel.BundleStartLevelDTO getBundleStartLevel(long id)
throws Exception

Throws:
Exception

getBundleStartLevel

org.osgi.dto.framework.startlevel.BundleStartLevelDTO getBundleStartLevel(String bundlePath)
throws Exception

Throws:
Exception
setBundleStartLevel

void setBundleStartLevel(long id,
    org.osgi.dto.framework.startlevel.BundleStartLevelDTO startLevel)
    throws Exception

Throws:
    Exception

setBundleStartLevel

void setBundleStartLevel(String bundlePath,
    org.osgi.dto.framework.startlevel.BundleStartLevelDTO startLevel)
    throws Exception

Throws:
    Exception

installBundle

String installBundle(String url)
    throws Exception

Throws:
    Exception

installBundle

String installBundle(String location, InputStream in)
    throws Exception

Throws:
    Exception

uninstallBundle

void uninstallBundle(long id)
    throws Exception

Throws:
    Exception

uninstallBundle

void uninstallBundle(String bundlePath)
    throws Exception

Throws:
    Exception

updateBundle

void updateBundle(long id)
    throws Exception

Throws:
    Exception
updateBundle

void updateBundle(long id,  
String url)  
throws Exception  
Throws:  
Exception

updateBundle

void updateBundle(long id,  
InputStream in)  
throws Exception  
Throws:  
Exception

getServices

Collection<String> getServices()  
throws Exception  
Throws:  
Exception

getServices

Collection<String> getServices(String filter)  
throws Exception  
Throws:  
Exception

getServiceRepresentations

Collection<org.osgi.dto.framework.ServiceReferenceDTO> getServiceRepresentations()  
throws Exception  
Throws:  
Exception

getServiceRepresentations

Collection<org.osgi.dto.framework.ServiceReferenceDTO> getServiceRepresentations(String filter)  
throws Exception  
Throws:  
Exception

ggetServiceReference

org.osgi.dto.framework.ServiceReferenceDTO getServiceReference(long id)  
throws Exception  
Throws:  
Exception
getServiceReference

org.osgi.dto.framework.ServiceReferenceDTO getServiceReference(String servicePath)

Throws:

Exception
Package org.osgi.rest.client

Interface RestClientFactory

org.osgi.service.rest.client
public interface RestClientFactory

Method Detail

createRestClient

RestClient createRestClient(URL url)
# REST Client Javascript

## 9.1.1 OsgiRestClient(baseUrl)

The OSGi REST API client

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseUrl</td>
<td>the base URL to the REST API.</td>
<td></td>
</tr>
</tbody>
</table>

Returns:
- the OSGi REST API client object

## 9.1.2 Methods

### 9.1.2.1 getBundle(b)

get the Bundle representation of a specific bundle

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>the bundle, either the numeric bundle ID or the bundle URI path.</td>
<td></td>
</tr>
</tbody>
</table>

Returns:
- the Bundle representation as a JSON object.

### 9.1.2.2 getBundlesRepresentations()

get the bundle representations of all bundles

Returns:
- a JSON array containing all Bundle representations

### 9.1.2.3 getBundles()

get the bundles

Returns:
- the URI paths of the bundles as a JSON array of strings

### 9.1.2.4 getBundleStartLevel(b)

Get the bundle startlevel.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>the bundle, either the numeric bundle ID or the bundle URI path.</td>
<td></td>
</tr>
</tbody>
</table>
9.1.2.5 `bundleState(b)`
Get the state of a bundle.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td></td>
<td>the bundle, either the numeric bundle ID or the bundle URI path.</td>
</tr>
</tbody>
</table>

Returns:
the bundle state representation.

9.1.2.6 `getFrameworkStartLevel()`
get the framework start level in JSON FrameworkStartLevel representation.

9.1.2.7 `getServiceRepresentations()`
Get the representations of all services

Returns:
a JSON array containing all representations.

9.1.2.8 `getServices()`
Get all services.

Returns:
a JSON array of the URI paths of all services.

9.1.2.9 `getServices(s)`
Get the representation of a service.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td></td>
<td>the service, either the numeric service ID or the service URI path.</td>
</tr>
</tbody>
</table>

Returns:
the service representation.

9.1.2.10 `installBundle(uri)`
Install a new bundle.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uri</td>
<td></td>
<td>the URI of the bundle to be installed</td>
</tr>
</tbody>
</table>

Returns:
the URI path of the newly installed bundle.

9.1.2.11 `setBundleStartLevel(b, the)`
Set the startlevel of a bundle.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td></td>
<td>the bundle, either the numeric bundle ID or the bundle URI path.</td>
</tr>
</tbody>
</table>
9.1.2.12 setBundleState(b, state)
Set the state of a bundle to start or stop it.

Parameters:

Name  | Type | Description
--- | --- | ---
 b   | the bundle, either the numeric bundle ID or the bundle URI path.
 state | the target state.

Returns:
the updated state of the bundle.

9.1.2.13 setFrameworkStartLevel(the)
set the framework start level

Parameters:

Name  | Type | Description
--- | --- | ---
 the | new framework startlevel in JSON representation.

Returns:
the updated framework startlevel in JSON representation.

9.1.2.14 startBundle(b)
Start a bundle.

Parameters:

Name  | Type | Description
--- | --- | ---
 b | the bundle, either the numeric bundle ID or the bundle URI path.

Returns:
the updated bundle state representation.

9.1.2.15 stopBundle(b)
Stop a bundle.

Parameters:

Name  | Type | Description
--- | --- | ---
 b | the bundle, either the numeric bundle ID or the bundle URI path.

Returns:
the updated bundle state representation.

9.1.2.16 uninstallBundle(b)
Uninstall a bundle.

Parameters:
Class RestClient

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td></td>
<td>the bundle, either the numeric bundle ID or the bundle URI path.</td>
</tr>
</tbody>
</table>

9.1.2.17 updateBundle(b, uri)
Update a bundle.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td></td>
<td>the bundle, either the numeric bundle ID or the bundle URI path.</td>
</tr>
<tr>
<td>uri</td>
<td></td>
<td>the URI from which to update the bundle.</td>
</tr>
</tbody>
</table>

10 Security Considerations

Like any externally visible management interface, the REST interface exposes privileged operations and hence requires access control. Since REST builds upon the HTTP(s) protocol, authentication mechanisms and encryption can be applied the same way as usually done for web servers: they can be layered below the REST protocol. E.g., confidentiality of the transmitted commands can be ensured by using HTTPS as the underlying transport. Authentication can be added by requiring, e.g., basic authentication prior to accepting a REST command.

The REST interface should only be implemented by a trusted bundle. Implementations of this specification require all admin permissions and all service permissions.

11 Document Support

11.1 References

[3]. RFP 133: Cloud Computing

11.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Jan S. Rellermeyer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM Research</td>
</tr>
<tr>
<td>Address</td>
<td>11501 Burnet Rd.</td>
</tr>
<tr>
<td></td>
<td>Austin, TX 78758</td>
</tr>
<tr>
<td>e-mail</td>
<td>relermeyer_at_us.ibm.com</td>
</tr>
</tbody>
</table>

11.3 Acronyms and Abbreviations

REST – Representational State Transfer
HTTP – Hypertext Transfer Protocol

11.4 End of Document
Abstract

The Computing Cloud often provides a highly dynamic environment where the load of a system might change, the topology of the cloud nodes might change or the requirements on the application may change at runtime. This document describes an OSGi cloud environment where nodes and capabilities can be discovered dynamically through OSGi Services and the deployment topology can be changed at runtime to react in changes in the observed characteristics, topology or requirements. An OSGi cloud can also be repurposed which can save network bandwidth as VM images only need to be sent to cloud nodes once.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGEMENT ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0 Document Information.........................................................................................................................................2
  0.1 License....................................................................................................................................................2
  0.2 Trademarks..............................................................................................................................................3
  0.3 Feedback...................................................................................................................................................3
  0.4 Table of Contents......................................................................................................................................3
  0.5 Terminology and Document Conventions...............................................................4
  0.6 Revision History......................................................................................................................................4

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

2 Application Domain......................................................................................................................................6
  2.1 Terminology + Abbreviations.................................................................................................................8

3 Problem Description.....................................................................................................................................9

4 Requirements................................................................................................................................................10

5 Technical Solution........................................................................................................................................11
  5.1 Platform requirements..............................................................................................................................12
  5.2 Framework Node Status Service.............................................................................................................13
    5.2.1 Obtaining dynamic framework metadata........................................................................................15
    5.2.2 Restrictions.....................................................................................................................................16
    5.2.3 Variable change notification.............................................................................................................16
  5.3 Service Distribution.................................................................................................................................17
  5.4 Remote Services Admin extensions.........................................................................................................17
6 Data Transfer Objects........................................................................................................ 21

7 Considered Alternatives..................................................................................................... 22
  7.1 Framework Discovery...................................................................................................... 22
  7.2 Service Discovery.......................................................................................................... 22
  7.3 Configuration Changes................................................................................................... 23
  7.4 Application-specific Framework-related metadata......................................................... 23

8 Security Considerations...................................................................................................... 23

9 Document Support............................................................................................................... 23
  9.1 References.................................................................................................................... 23
  9.2 Author's Address........................................................................................................... 24
  9.3 Acronyms and Abbreviations........................................................................................ 25
  9.4 End of Document........................................................................................................... 25

0.5 Terminology and Document Conventions
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>August, 2011</td>
<td>David Bosschaert, Initial Version</td>
</tr>
<tr>
<td>0.2</td>
<td>April, 2012</td>
<td>Richard Nicholson, Additional Input</td>
</tr>
<tr>
<td>0.3</td>
<td>April, 2012</td>
<td>David Bosschaert, Prepare for F2F</td>
</tr>
<tr>
<td>0.4</td>
<td>October, 2012</td>
<td>David Bosschaert, incorporate feedback from Basel F2F, introduce Ecosystems.</td>
</tr>
<tr>
<td>0.5</td>
<td>November, 2012</td>
<td>Marc Schaaf, Minor changes and some comments for Orlando F2F.</td>
</tr>
<tr>
<td>0.6</td>
<td>November, 2012</td>
<td>David Bosschaert, incorporate feedback from Orlando F2F.</td>
</tr>
<tr>
<td>0.7</td>
<td>December, 2012</td>
<td>Richard Nicholson – feedback / comments</td>
</tr>
<tr>
<td>0.8</td>
<td>January, 2013</td>
<td>David Bosschaert, some additional comments and clarification</td>
</tr>
</tbody>
</table>
1 Introduction

1st generation 'public Cloud' solutions have since their inception been built upon two enabler technologies

• Service orientation (increasingly REST centric) allowing dynamic find / bind / use of deployed Services & Resources.

• The virtual Machine - this used as the mechanism to:
  ○ Partition physical compute resource.
  ○ Via the virtual machine image - provide a standard deployment artifact; a static opaque software blob.

However, it is increasingly accepted that deployment of opaque virtual machine images consumes unnecessary network bandwidth and storage. As the dependencies are not understood – such approaches have larger downstream maintenance implications. For highly centralized / monolithic Cloud offerings – brute force infrastructure approaches - at significant capital cost – are possible (e.g. Amazon / Google offerings). However the dependency / maintenance issue is not addressed. Recent trends involving the deployment of software artefacts into a Cloud environment (Puppet, Chef) are a step in the right direction; but the software components remain coarse grained with non standard approaches to dependency management and configuration of the deployed artifacts. Industry standards bodies are retrospectively attempting to standardize topology and life-cycle for traditional applications via initiatives such as OASIS TOSCA & CAMP.
Meanwhile the next generation of Cloud will be driven by the edge – meaning pervasive / federated cloud solutions with service components running in a variety of environments including: 3rd generation mobile and home networks and more federated Cloud cores. For such environments only the minimum necessary software required to realize the Cloud service should be deployed, as required, to the appropriate location / device. Updates likewise limited to the necessary changes.

To achieve this, software modularity, sophisticated 'requirements 'and 'capabilities' driven dependency resolution and assembly - are the key enablers. Hence OSGi is uniquely placed within the industry to realize this vision; this especially so given the resurgence of Universal OSGi activities.

One of the key aspects of Cloud Computing is the fact that Resources are deployed on (virtual) machines, nodes, which are not pre-determined. When working with more complex cloud systems, where various components are deployed across different nodes, these components need to be connected with each other to form a working solution. Furthermore frequent changes in these deployments are common for such environments to allow dynamic scalability which again requires the discovery of newly added or removed components even after the initial deployment is finished.

Building upon the background research presented in RFP 133 [3], this RFC explores the intersection between OSGi and Cloud with specific emphasis on discovery, configuration and 'wire-up' of multi-node OSGi based applications in a dynamic Cloud environment through the use of OSGI APIs and mechanisms.

2 Application Domain

This RFC relates to Cloud Computing domains and use-cases but can also be useful in non-cloud environments. Cloud Computing is to a certain degree a marketing term and many of its concepts are applicable where distributed computing is used.

This RFC aims at providing a baseline platform that can address use cases around discovery of OSGi frameworks and other resources, provisioning and re-provisioning of deployables and reacting to change in the system by providing the primitives to discover and monitor the topology of a system. Combined with a remote deployment mechanism and utilizing the dynamic capabilities of the OSGi framework this provides the capabilities to control cloud deployments and change their characteristics at runtime.

The dynamic aspects of OSGi frameworks and its Service Registry map quite naturally to the dynamic behavior of Cloud systems where deployments may need to be modified during operation because of changes in demand or the running environment. The small footprint of OSGi frameworks themselves and the fact that they can be highly customized to the task at hand also fits well with Cloud scenarios where memory, computing power and storage facilities are often constrained or charged for per usage.

**OSGi Services and Distributed Systems**

Currently we have three categories of OSGi Service that are / or may potentially be / distributed in some form: namely RSA, Event Admin and ConfigAdmin. However there is no coherent / over-arching 'distributed architecture' which encompasses all, and explains the inter-relationship between, these OSGi services.
This is highlighted when one considers 'discovery'.

In addition to the type of entity being discovered (a RPC Service, message source/sink, REST resource etc) we MUST also consider 'change' as entities will usually have a mixture of static, slowly and rapidly changing properties.

It is important to make the distinction between occasionally changing properties and rapidly changing properties as different ways to advertise these may be appropriate.

**Discovery**

The following 'discovery' use cases are suggested

1) Resources Discovery – Available OSGi Frameworks in the specified environment; also physical resource in the environment which might act as a host for an OSGi Framework.

   Static properties might include attributes and capabilities such as location; ownership; access to other type of resource – i.e. required data, CPU/number of cores, OS, JVM or OSGi framework Type (all could be considered immutable).

   Dynamic properties attributes include installed bundles / sub systems at each point in time, current available memory, current load etc.

2) Deployed Artifacts - Artifacts of a particular type that have been deployed into the environment. Usually a subset of discovered Resources.

   Static properties might include name of bundle / sub-System, static configurations, requirements and capabilities.

   Dynamic properties might include resource consumption / performance metrics (monitoring), configurations which are dynamically configurable at runtime.

3) Available Services – Services that are available within the environment. Usually a subset of discovered / deployed artifacts.

   Static: Properties of the host – including all – or a subset of - resource and artefact 'Capabilities'

   Dynamic: Usually relating to Service performance – resource load, memory – number of items to be processed, reliability / quality metrics.

Note that (2) and (3) may be expressed as attributes of (1). However one might also wish to independently discover entities of a particular type.

**Remote Service Administration / Remote Services**

Distributed Service discovery is covered within the OSGi Alliance's Remote Service Administration specification.

Service endpoints with associated properties (service.properties) may be advertised via a pluggable discovery mechanism.

However, the specification is a little vague as to the nature of advertised service properties. Being in effect the local service.properties registered in the local service registry – are Service endpoint properties assumed to be almost immutable; i.e. infrequently changing. Current RSA implementations work this way – in that – if Service properties change – the 'old' service is removed and the 'new' service discovered.
If static / immutable properties change - then by definition - the new entity is no-longer the same as the one initially discovered. However, properties that relate to Services might be highly volatile; perhaps local performance statistics that we use to priority select / or balance across / service instances. If tracking a remote service and the properties change, we don't want to have to handle the service apparently disappearing and reappearing, especially if this happens frequently e.g. for properties added by the middleware such as “current load”.

Should these be treated as property ‘updates’ to those initial advertised? So under the ‘discovery’ umbrella. Or treated as a separate monitoring concerns? Perhaps –

- advertised properties used for discovery are immutable – but included in this static advertisement is the information required to subsequently subscribe to / receive volatile / dynamic property updates.

- Or, perhaps mutable / immutable properties remain collocated - defined in a more sophisticated service advertisement.s.

Note that while an underlying implementation is not defined - service ‘discovery’ is already event centric. Given this one might like to treat the processing of update / monitoring events differently to discovery events - but transported by the same distributed eventing mechanisms.

**Contracts Based Interactions**

It has already been suggested that, w.r.t. Interactions, we need to focus on the *contract*. Each participant has its own role with respect to the contract: i.e. what it is expected to provide and what it can expect from the other participants.

A contract is just the agreed set of interactions between modules. When we think about services we tend to think of a contract as a Java Interface... e.g. the CreditCheck service provides method calculateRating(). But that's a simplistic contract with one provider (i.e. the credit rating provider) and some consumers, and it doesn't *appear* to support asynchronous interaction because invocation always originates from a consumer.

But if you consider a group of interfaces then you have something more powerful, because each participant can provide some interfaces and consume some interfaces. For example a stock exchange: the exchange itself provides the OrderEntry, and other participants provide ExecutionListeners or MarketDataProviders or whatever. Hence, the ‘contract’ this not with a single Java interface but with a coherent collection of interfaces: in other words a package: the ‘contract’ concept spanning synchronous, asynchronous and event based interactions.

---

**2.1 Terminology + Abbreviations**

This document uses terms defined in OSGi RFP 133 Cloud Computing. The terms are based on the NIST definitions for Cloud Computing and common industry naming practice.

Additional terminology:

Cloud Ecosystem – a dynamic Cloud System.
Cloud-based applications are often composed of multiple components each running on one or more cloud nodes. For instance the following is an example application architecture.

![Diagram of a possible application architecture]

This example e-commerce application has a web front-end, a database, a message queue and a back-end component. Each of these components is replicated on various nodes. In order to function the components of the application need to know where (e.g. on what IP) other components can be found. Components also need to be kept informed of the liveness of their component dependencies.

In traditional deployments this kind of information is often kept in static files, handled via a hardware load-balancer or through a proprietary HA solution.

In a cloud scenario a standards-based solution is needed to enable the discovery of application components in a this dynamic environment.

For more context please refer to the Problem Description section in RFP 133 [3].
4 Requirements

This RFC covers the following requirements listed in RFP 133 [3].

MAN0004 – The solution MUST provide APIs to discover available OSGi Frameworks in a Resource Domain.

MD0001 – The solution MUST define APIs that allow querying of capabilities and other metadata of OSGi Frameworks in the Cloud. This information SHOULD at least include the following:

- Framework GUID

MD0002 – An OSGi bundle MUST be able to add proprietary capabilities to the metadata exposed by its OSGi Framework in the Cloud.

MD0003 – The solution MUST provide information about the environment, system, and the capabilities and properties of the platform underlying an OSGi Framework in the Cloud. This information SHOULD include at least the following static capabilities:

- location
- IP address
- cpu architecture
- cpu capacity
- Total memory

And the following dynamic capabilities

- cpu load factor
- Available Memory

MD0004 – The solution MUST allow provider-specific capabilities to be added regarding the underlying platform.

MD0007 – The solution MUST allow querying the available OSGi Frameworks in a Cloud Domain based on the metadata and capabilities these OSGi Frameworks expose.

Additional requirements obtained during the EclipseCon 2012 Cloud Workshop:

CWS0010 – Make it possible to describe various Service unavailable States. A service may be unavailable in the cloud because of a variety of reasons.

- Maybe the amount of invocations available to you have exhausted for today.
- Maybe your credit card expired
• Maybe the node running the service crashed.

It should be possible to model these various failure states and it should also be possible to register ‘callback’ mechanisms that can deal with these states in whatever way is appropriate (blacklist the service, wait a while, send an email to the credit card holder, etc).

CWS0020 – Come up with a common and agreed architecture for Discovery. This should include consideration of Remote Services, Remote Events and Distributed Configuration Admin.

CWS0030 – Resource utilization. It should be possible to measure/report this for each cloud node. Number of threads available, amount of memory, power consumption etc…

CWS0040 – We need subsystems across frameworks. Possibly refer to them as 'Ecosystems'. These describe a number of subsystems deployed across a number of frameworks.

CWS0050 – It should be possible to look at the cloud system state:

• where am I (type of cloud, geographical location)?
• what nodes are there and what is their state?
• what frameworks are available in this cloud system?

| where’s my service in the cloudOBR?
| what state am I in?
| what do I need here in order to operate?
| etc…

CWS0060 – Deployment - when deploying replicated nodes it should be possible to specify that the replica should “not” be deployed on certain nodes, to avoid that all the replicas are deployed on the same node.

## 5 Technical Solution

A complete solution can be broken into the following considerations:

1. Definition of functional and deployment topologies – for System / EcoSystem
2. Discovery of Resource
3. Method of deployment – mapping required topology to available resource.
4. Subsequent re-discovery of deployed artifacts / available resource

5. Interaction models between deployed components.

This specification describes a platform where multiple OSGi Framework instances are running in different Java VMs and often on different actual or virtual computing nodes.

The platform provider provides the Ecosystem administrator with tools to create new computing nodes and to associate these nodes with an ecosystem. Each node is associated with at most one ecosystem, while a single ecosystem can be associated with many nodes. Note however that a cloud node may be virtual, so multiple nodes could potentially be hosted within the same platform or infrastructure through multi-tenancy.

This specification does not describe the how the compute nodes are created and associated with the ecosystem. Platform providers can provide proprietary solutions for this which may be realized through a web-based admin console, a set of command-line utilities, a REST-based API or otherwise.

This specification describes a means of discovering the topology of OSGi nodes in the ecosystem and provides primitives to provision these, which includes their initial provisioning as well as applying ongoing changes to the provisioning of the system during runtime, to react to changes in the ecosystem topology as well as changes in the runtime characteristics of the application, i.e. to scale up or scale down dynamically.

Therefore the scope of interest is 1, 2, 4 & 5 from the above list.

While concern 3 in the above list can be realized through the primitives in this specification, this document does not describe a format to declare a mapping from a topology to resources. It is expected that this will be done in a separate RFC.

5.1 Platform requirements

Platforms compliant with this specification provide the following components on each computing node.
1. An OSGi Core Framework as defined by the Core R6 specification.

2. A Remote Services Distribution Provider that understands the osgi.configtype.ecosystem Configuration Type.

3. A discovery mechanism providing visibility to all remoted OSGi services with the osgi.configtype.ecosystem Configuration Type to all other OSGi frameworks in the same ecosystem.

4. A component which registers an FrameworkNodeStatus service for each Framework running in the ecosystem. These FrameworkNodeStatus objects are registered as remote services with the osgi.configtype.ecosystem configuration type.

5. A remote management mechanism compliant with RFC 182. This mechanism should be accessible from within the ecosystem, but does not need to be accessible from the outside world.

### 5.2 Framework Node Status Service

The FrameworkNodeStatus service advertises the availability of an OSGi framework in the Ecosystem. The Framework can be exported with the Remote Services configuration type osgi.configtype.ecosystem so that other frameworks running in the same ecosystem have visibility of this Framework. However alternative mechanisms to distribute this service across the Ecosystem are also permitted. Furthermore, other frameworks can listen for services of this type to appear, disappear or change if they wish to be notified of changes occurring in the ecosystem. This is vital information for a Management agent or Provisioning component and can also be used to dynamically re-scale and re-purpose the deployments in the ecosystem.
Only nodes using the Service will get metadata mirrored

The FrameworkNodeStatus service provides metadata about the framework via Service Registration properties as well as via variables which can be obtained via the getVariable() API. Static or mostly static metadata is represented as Service Registration properties, dynamic metadata is represented as variables.

The FrameworkNodeStatus is available with the following Service Registration Properties.

<table>
<thead>
<tr>
<th>key</th>
<th>data type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.framework.uuid</td>
<td>String</td>
<td>The globally unique ID for this framework.</td>
</tr>
<tr>
<td>org.osgi.node.hostip</td>
<td>String+</td>
<td>The external host names or IP addresses for this OSGi Framework, if exists.</td>
</tr>
<tr>
<td>org.osgi.node.hostip.internal</td>
<td>String+</td>
<td>The internal host names or IP addresses for this OSGi Framework for access from inside the Ecosystem, if exists.</td>
</tr>
<tr>
<td>org.osgi.node.type</td>
<td>String</td>
<td>The name of the Cloud/Environment in which the Ecosystem operates.</td>
</tr>
<tr>
<td>org.osgi.node.version</td>
<td>String</td>
<td>The version of the Cloud/Environment in which the Ecosystem operates.</td>
</tr>
<tr>
<td>org.osgi.node.country</td>
<td>String</td>
<td>ISO 3166-1 alpha-3 location where this Framework instance is running, if known.</td>
</tr>
<tr>
<td>org.osgi.node.location</td>
<td>String</td>
<td>ISO 3166-2 location where this framework instance is running, if known. This location is more detailed than the country code as it may contain province or territory.</td>
</tr>
<tr>
<td>org.osgi.node.region</td>
<td>String</td>
<td>Something smaller than a country and bigger than a location (e.g. us-east) ??? Jan Rellermeyer to provide more details...</td>
</tr>
<tr>
<td>org.osgi.node.rest.url</td>
<td>String (URL, optional)</td>
<td>The external URL of the framework management REST API, if available.</td>
</tr>
<tr>
<td>org.osgi.node.rest.url.internal</td>
<td>String (URL, optional)</td>
<td>The ecosystem-internal URL of the framework management API, if available.</td>
</tr>
<tr>
<td>org.osgi.framework.version</td>
<td>String (URL, optional)</td>
<td>The value of the Framework properties as obtained via</td>
</tr>
</tbody>
</table>
### 5.2.1 Obtaining dynamic framework metadata

While service registration properties are used to advertise and obtain static or mostly static metadata, dynamic metadata is available via the FrameworkNodeStatus.getVariable() API. Invoking this API will generally cause a remote service invocation to be performed in order to obtain the current value of the variable requested.

The following variable names are predefined, however none of these are required to be supported:

<table>
<thead>
<tr>
<th>variable name</th>
<th>data type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>available.memory</td>
<td>Long</td>
<td>The amount of memory available to the JVM in which the Framework runs in kilobytes.</td>
</tr>
<tr>
<td>available.diskspace</td>
<td>Long</td>
<td>The amount of disk space available to the JVM where the Framework runs in kilobytes. This could be temporary disk space that does not persist across node restarts.</td>
</tr>
<tr>
<td>processor.load</td>
<td>Integer [0..100]</td>
<td>The load of the machine. 0 means no load at all where 100 means operating at full capacity.</td>
</tr>
</tbody>
</table>

The getVariable() API returns all values as a String. The data type above provides guidance on the interpretation of the value.

### 5.2.2 Application-specific FrameworkNodeStatus metadata

Bundles can register a local FrameworkNodeAddition Service (via the Whiteboard pattern) that allows them to provide additional metadata associated with the FrameworkNodeStatus service. This allows the build-up of a catalog of capabilities, which is visible to remote frameworks but can also be of use in the local framework.

The FrameworkNodeAddition service is registered with the following properties:
### Property Data Type Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add.properties</td>
<td>String+</td>
<td>The names of additional framework properties. The specified properties will be copied from this service registration to the FrameworkNodeStatus service registration that is available to remote frameworks.</td>
</tr>
<tr>
<td>add.variables</td>
<td>String+</td>
<td>The names of additional framework variables. The variables can be accessed through the getVariable() API and are included in the listVariableNames() API.</td>
</tr>
</tbody>
</table>

When multiple FrameworkNodeAddition services provide the same property or variable, the service with the highest service.ranking wins. In case of a tie, the service with the lowest service.id wins.

The service has the following API:

```java
public interface FrameworkNodeAddition {
    String getVariable(ClientContext client, String name);
}
```

When clients invoke the FrameworkNodeStatus.getVariable() API for the added variables, the FrameworkNodeAddition service will receive a callback to handle the invocation. Note that the implementation is permitted to return different values for a given variable for different clients. For example clients with a different SLA can be provided with different values for a certain variable. The implementation can use the ClientContext passed in to distinguish between clients.

For example the following adds a service property `my-app.role=database` and a framework variable `network.load` to the FrameworkNodeStatus service:

```java
Dictionary<String, Object> d = new Hashtable<String, Object>();
d.put("add.properties", new String [] {"my-app.role");
d.put("my-app.role", "database");
d.put("add.variables", "network.load");
ctx.registerService(FrameworkNodeAddition.class.getName(),
    new MyAdditionImpl(), d);
```

The implementation of the FrameworkNodeAddition interface provides a callback mechanism to provide the network.load variable:

```java
public class MyAdditionImpl implements FrameworkNodeAddition {
    @Override
    public String getFrameworkVariable(String name, ClientContext client) {
        if ("network.load".equals(name)) {
            return networkInfo.getLoad()
        } else {
            throw new IllegalArgumentException(name);
        }
    }
}
```

#### 5.2.3 Restrictions

Added property or variable names may not start with the following prefixes: “org.osgi.”, “osgi.”, “java.”, “objectClass” or “service.”.

#### 5.2.4 Variable change notification

Consumers can subscribe to change notifications for Framework variables via the OSGi Event Admin Service. To obtain these notifications across VM boundaries a Distributed OSGi Event Admin can be used, see RFC XXX.
5.3 Service Distribution

OSGi Remote Services functionality is available throughout the ecosystem via the osgi.configtype.ecosystem configuration type. This configuration type can make OSGi services remotely available that restrict their API as follows:

- The services are registered under one interface only.
- The methods parameters and return types of the interface are restricted to basic datatypes as used for service properties: primitive types, their wrappers as well as arrays and basic collections (List, Set and Map).
- Additionally, method parameters and return types can be also be of interfaces composed using the previous restrictions.

Given any service that is implemented using these restrictions, registering the service with the interface specified in the service.exported.interfaces and the osgi.configtype.ecosystem as the service.exported.configs service registration property will make the service available to service consumers in all frameworks within the ecosystem.

5.4 Remote Services Admin extensions

The Remote Services Admin specification needs to be expanded.

- The EndpointListener interface in this specification was missing a modification callback. This needs to be added.
- The remote service provider side needs to gain more control over how remote clients invoke the service. For example it may need to do some bookkeeping and count every time a client invokes the service. Or it may decide to give every client a unique instance of the service, if this service holds state for that client.
- Remote services need to be able to provide metadata about themselves. This may include being able to communicate to clients whether the service is operating normally, whether there is maintenance being planned, whether payment or other quota-related issues cause disruption or otherwise.

5.4.1 Providing control over remote clients

To provide control over how clients can invoke the service, the service can be registered via a RemoteServiceInvocationHandler. The RemoteServiceInvocationHandler allows the implementor to:

- Identify the client. This is done via the ClientContext object.
- Count the client invocations.
- Block client invocations.
- Decide which backing object to use to handle the invocation. This allows pooling of backing objects in cases where each object can only be used by one client at a time. It also allows handing out of separate objects for each client or even separate objects for each invocation.

The interface of the RemoteServiceInvocationHandler is as follows:

```java
/**
 * Provides control over services used by remote clients. When an instance of this
 */
```
* class is attached to the "service.exported.handler" property of the service registration the handler will be called for each remote invocation instead of the service object directly.

```java
public interface RemoteServiceInvocationHandler<T> {

    /**
     * Invoke the service on behalf of a remote client.
     * @param client Information in relation to the client.
     * @param reference The Service Reference representing the service being invoked.
     * @param method The method of the service being invoked.
     * @param args The method arguments.
     * @return Provide the return value to be returned to the remote client.
     * @throws Exception if the embedded invocation throws an exception.
     */
    public Object invoke(ClientContext client, ServiceReference<T> reference, Method method, Object[] args) throws Exception;
}
```

The ClientContext interface provides information about the remote client which is making the invocation. It is defined as follows, some of the methods may return `null` as not all the information may be available all the time:

```java
public interface ClientContext {
    String getHostIPAddress();
    String getFrameworkUUID();

    SubjectPrincipal getSubjectPrincipal();
    Map<String, Object> getProperties();
}
```

The following is an example implementation of a RemoteServiceInvocationHandler that disallows concurrent invocations by the same remote client:

```java
public class MyServiceInvocationHandler implements RemoteServiceInvocationHandler<MyService> {
    ConcurrentMap<String, Object> activeClients = new ConcurrentHashMap<String, Object>();

    @Override
    public Object invoke(ClientContext client, ServiceReference reference, Method method, Object[] args) {
        if (activeClients.putIfAbsent(client, Boolean.TRUE) != null)
            throw new IllegalStateException("Only 1 concurrent invocation allowed per client.");
    }
}
try {
    return method.invoke(new MyServiceImpl(), args);
} finally {
    activeClients.remove(client);
}

The remote service MyService is then registered as follows:

```java
Hashtable<String, Object> props = new Hashtable<String, Object>();
props.put("service.exported.interfaces", "*");
props.put("service.exported.configs", new String [] {
    "osgi.configtype.ecosystem", "<<nodefault>>"});
props.put("service.exported.handler", new MyServiceInvocationHandler());
bundleContext.registerService(MyService.class, new MyService(), props);
```

### 5.4.2 Remote Service Metadata

Remote services can fail for reasons generally not applicable to local services. Sometimes the reasons for failure are non-technical. Maybe the credit card used for payment has expired, or maybe the service is down for maintenance.

#### 5.4.2.1 Obtaining Remote Service Metadata

A Remote Service can provide metadata can be obtained around its availability or other concern through the RemoteServiceMetadataProvider Service API. This metadata can be instrumental in deciding which service instance to invoke if more than one candidate is available. It can also be useful in preventing operational failure or in diagnosing failure scenarios. The RemoteServiceMetadataProvider Service has the following API:

```java
/**
 * Provide access to metadata, such as Service Variables of Remote Services.
 *
 * @param sref The Service Reference
 * @return The Remote Service Metadata for the service or null if there
 * no metadata for this service or it does not reference a remote service.
 */
RemoteServiceMetadata getMetadata(ServiceReference sref);
```

```java
/**
 * Obtain Remote Service Metadata for a service denoted by an endpoint ID.
 *
 * @param endpointID The endpoint ID as advertised via the endpoint.id service property of a remote service.
 * @return The Remote Service Metadata for the endpoint or null if no
 * metadata for this service can be found.
 */
```
RemoteServiceMetadata getMetadata(String endpointID);

/**
 * Obtain Remote Service Metadata for a service denoted by an endpoint ID and framework UUID. This variant can be useful if endpoint IDs are not unique across frameworks.
 * @param endpointID The endpoint ID as advertised via the {@code endpoint.id} service property of a remote service.
 * @param endpointFrameworkUUID The framework uuid as advertised via the {@code endpoint.framework.uuid} property.
 * @return The Remote Service Metadata for the endpoint or null if no metadata for this service can be found.
 */
RemoteServiceMetadata getMetadata(String endpointID, String endpointFrameworkUUID);
}

The RemoteServiceMetadata object obtained gives access to remote service variables:

public interface RemoteServiceMetadataProvider {
    String[] listVariablesNames();
    String getVariable(String name);
    Map<String, String> getVariables(String ... filter);
}

The Service Registration of the RemoteServiceMetadataProvider Service also provides access to the service.status variable of all tracked remote services, if this variable is provided by the service. For example one could find the following service registration properties on a RemoteServiceMetadataProvider service:

<table>
<thead>
<tr>
<th>service.id</th>
<th>67</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectClass</td>
<td>[org.apache.cxf.dosgi.dsw. _RemoteServiceMetadataProvider]</td>
</tr>
<tr>
<td><a href="http://acme.org:7711/63:30ca2d2f-7dfb-0012-19cb-8be6a8493417">http://acme.org:7711/63:30ca2d2f-7dfb-0012-19cb-8be6a8493417</a></td>
<td>&quot;OK&quot;</td>
</tr>
<tr>
<td><a href="http://acme.org:7711/63:90d8cf28-7dfb-0012-1e73-d36fd3a02552">http://acme.org:7711/63:90d8cf28-7dfb-0012-1e73-d36fd3a02552</a></td>
<td>&quot;WARNING&quot;</td>
</tr>
<tr>
<td><a href="http://acme.org:7711/51:30ca2d2f-7dfb-0012-19cb-8be6a8493417">http://acme.org:7711/51:30ca2d2f-7dfb-0012-19cb-8be6a8493417</a></td>
<td>&quot;PAYMENT_NEEDED&quot;</td>
</tr>
</tbody>
</table>

Remote services are represented as service properties in the following format: [endpoint id]:[endpoint framework uuid].

Through the service registration properties a client can register a simple service listener to get notified of changes in the service.status variable of a remote service.

The Remote Services implementation should ensure that client-side proxies of services that have a status value other than OK or WARNING are unregistered. These remote services should still be represented on the RemoteServiceMetadataProvider service to allow clients to find out why the service is in the error state and hence the proxy has disappeared.

Note that the RemoteServiceMetadataProvider service itself is not necessarily a remote service. It could be registered locally by the Remote Services implementation. The metadata can be obtained by a
consumer through the "service.imported.metadata" property of an imported remote service:

```java
Object mdp = sref.getProperty("service.imported.metadata");
if (mdp instanceof RemoteServiceMetadataProvider) {
    RemoteServiceMetadataProvider provider = (RemoteServiceMetadataProvider) mdp;
    System.out.println(provider.getVariable("remaining.invocations"));
}
```

5.4.2.2 Providing Remote Service Metadata

Remote Service Metadata is always specific to the service and must therefore be plugged in by the service provider. When a remote service is registered a handler object can be provided. This handler object can provide the service metadata as well as handle the actual remote invocations (see section 5.4.1). Remote Service Metadata is provided through an instance of the RemoteServiceMetadataHandler interface:

```java
public interface RemoteServiceMetadataHandler {
    String[] listVariableNames(ClientContext client);
    String getVariable(ClientContext client, String name);
}
```

An implementation of this interface can be registered with the "service.exported.handler" property alongside the other Remote Service related properties, for example:

```java
TestService ts = new TestServiceImpl();
Dictionary<String, Object> props = new Hashtable<String, Object>();
props.put("service.exported.interfaces", "*");
props.put("service.exported.configs", new String[] {
    "osgi.configtype.ecosystem", "<<nodefault>>"});
props.put("service.exported.handler", new TestServiceMetadataHandler());
context.registerService(TestService.class.getName(), ts, props);
```

The Remote Service's implementation ensures that the handler provided in such a way handles calls from remote clients make through the RemoteServiceMetadataProvider Service on the object obtained from the "service.imported.metadata" property on the client-side service reference.

Note that to provide both an invocation handler as well as a metadata handler a single object implementing both the RemoteServiceInvocationHandler as well as the RemoteServiceMetadataHandler interface can be used.

5.4.2.3 Service variables

The following variable names are predefined but none of these are mandatory:
<table>
<thead>
<tr>
<th>variable name</th>
<th>data type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service.status</td>
<td>String, from:</td>
<td>Returns the status of the service. Return values that start with OK mean that the service can be invoked. Other return values indicate that service invocation will result in an error.</td>
</tr>
<tr>
<td></td>
<td>&quot;OK&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;PAYMENT_NEEDED&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;UNAUTHORIZED&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;FORBIDDEN&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;NOT_FOUND&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;QUOTA_EXCEEDED&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;SERVER_ERROR&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;TEMPORARY_UNAVAILABLE&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;WARNING&quot;</td>
<td></td>
</tr>
<tr>
<td>service.detail</td>
<td>&quot;QUOTA REMAINING:&lt;int&gt;[:&lt;reset-date&gt;]&quot;</td>
<td>If the service.status has the value other than OK this variable can be used to obtain machine-readable details.</td>
</tr>
<tr>
<td></td>
<td>&quot;PAYMENT_NEEDED_AT:&lt;date&gt;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;TEMPORARY_UNAVAILABLE:&lt;date&gt;:&lt;duration-mins&gt;&quot;</td>
<td></td>
</tr>
<tr>
<td>... custom ...</td>
<td></td>
<td>Implementations may provide custom variables as appropriate.</td>
</tr>
</tbody>
</table>

5.4.3 EndpointListener enhancement

This section has moved to bug 164: [https://www.osgi.org/bugzilla/show_bug.cgi?id=164](https://www.osgi.org/bugzilla/show_bug.cgi?id=164)

6 Data Transfer Objects

RFC 185 defines Data Transfer Objects as a generic means for management solutions to interact with runtime entities in an OSGi Framework. DTOs provides a common, easily serializable representation of the technology.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a DTO? The expectation is that in most cases it would.

The DTOs for the design in this RFC should be described here and if there are no DTOs being defined 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.
7 Considered Alternatives

7.1 Framework Discovery

It needs to be possible to discover the available OSGi frameworks in the cloud system. Along with the existence of the frameworks themselves it must be possible to discover metadata about the frameworks, such as machine characteristics, cloud provider and other cloud metadata and information such as the physical location of the machine (country) and the IP address of the machine.

This information can be used to make provisioning-based decisions but also to configure services available in the cloud system that are not directly represented as distributed OSGi service, for example a Messaging System which needs to be configured with an IP address of the broker.

Potential ways to realize this:

- Via an RSA-distributed service that represents an OSGi Framework.
  - This requires that RSA will be expanded to support update of Service properties, as framework characteristics change at runtime.
  - pros: strong support for services in the framework. Ability to reuse LDAP service queries to find matching frameworks.
  - cons: ?

- Via an eventing mechanism
  - Paremus has some experience with this – TODO need to elaborate the potential design.

- Through a Repository API that represents Frameworks in the System. In this case the contents of the repository is all the nodes in the Cloud System.
  - A Repository (-like) API can be used to select frameworks in the Cloud System.
  - pros: use Generic Capabilities and requirements to select matching frameworks.
  - cons: Generic Capabilities and requirements are less suitable for dynamically changing properties.

7.2 Service Discovery

Remoted OSGi Services must be discoverable in the Cloud System. RSA-based distribution is the most suitable realization for this, however it must be expanded to cover service property changes.
7.3 Configuration Changes

OSGi Configuration Admin is a highly versatile dynamic configuration system, which should be suitable for usage in a Cloud System. The Configuration Consumption API is already suitable for remote distribution but we should look into whether the Administration API needs to be enhanced. Is it necessary to be able to target a specific cloud node via Configuration Admin? And if so how can that be realized?

7.4 Application-specific Framework-related metadata

Introduce an OSGiFrameworkPublisher service that allows applications to add/remove/modify service registration properties on the (remoted) OSGiFramework service.

Note the properties may not start with org.osgi., java. or service.

TODO describe this service API

TODO the API should also allow for the registration of new framework and service variables

8 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.

9 Document Support

9.1 References


# 9.2 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>Richard Nicholson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Paremus</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Voice</td>
<td></td>
</tr>
<tr>
<td>e-mail</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Marc Schaaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td></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:marc.schaaf@fhnw.ch">marc.schaaf@fhnw.ch</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Steffen Kächele</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>University of Ulm</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:steffen.kaechele@uni-ulm.de">steffen.kaechele@uni-ulm.de</a></td>
</tr>
</tbody>
</table>
9.3 Acronyms and Abbreviations

9.4 End of Document
Abstract

Update Blueprint Container to 1.1 with some features requested via bug reports or directly from users.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree not to 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, sublicensable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS Delineated Above.

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0 Document Information.........................................................................................................................2
  0.1 License........................................................................................................................................2
  0.2 Trademarks.................................................................................................................................3
  0.3 Feedback.......................................................................................................................................3
  0.4 Table of Contents.........................................................................................................................3
  0.5 Terminology and Document Conventions...................................................................................4
  0.6 Revision History..........................................................................................................................4

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

2 Application Domain..........................................................................................................................5

3 Problem Description..........................................................................................................................6
  3.1 3.1 Non-Damped Reference Managers.........................................................................................6
  3.2 3.2 Blueprint Grace Timeout........................................................................................................6
  3.3 3.3 Bug 2233 –‘Satisfied’ Lifecycle Notification..........................................................................6
  3.4 3.4 Injection of Service Properties...............................................................................................6
  3.5 3.5 Bug 2192 - Damping of Factory Services..............................................................................7
  3.6 3.6 Bug 1295 - Allow Namespace handlers to use inline Blueprint elements..........................7
  3.7 3.7 Bug 2406 – Add blueprint extender capability definition......................................................7
  3.8 Bug 2484 – Address lack of Blueprint opt-in header..................................................................7
  3.9 3.8 Allow service properties to be added via <property> elements in Service Manager............8
3.10 3.9 Add synchronous start mode for Blueprint Container..................................................8

4 Requirements..................................................................................................................................................8

5 Technical Solution ......................................................................................................................................9
  5.1 Non-Damped Reference Managers ........................................................................................................9
  5.2 Blueprint Grace Period enhancements .....................................................................................................9
  5.3 New Satisfied Life-cycleEvent ..................................................................................................................10
  5.4 Injection of Service Properties ...............................................................................................................10
  5.5 Factory Services .......................................................................................................................................10
  5.6 Inlining of Blueprint elements ................................................................................................................10
  5.7 Extender Capability ................................................................................................................................10
  5.8 Allow <property> elements in Service Managers .................................................................................11
  5.9 Add synchronous start mode for Blueprint Container ..........................................................................11
  5.10 Blueprint Extender Configuration (Bug 2484) .........................................................................................11
      5.10.1 Extendee Header Behavior .............................................................................................................11
      5.10.2 TODO: Other things configured through the dictionary ..............................................................12

6 Considered Alternatives ..........................................................................................................................12

7 Security Considerations ..........................................................................................................................12

8 Document Support .....................................................................................................................................12
  8.1 References ...............................................................................................................................................12
  8.2 Author’s Address ......................................................................................................................................13
  8.3 Acronyms and Abbreviations ................................................................................................................13
  8.4 End of Document .....................................................................................................................................13

0.5 Terminology and Document Conventions
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>21/05/12</td>
<td>Initial Draft Requirements</td>
</tr>
<tr>
<td>0.2</td>
<td>2nd November 2012</td>
<td>First stab at design.</td>
</tr>
<tr>
<td>0.3</td>
<td>27th February</td>
<td>Updates from Nov 2f2 &amp; tidy-up for draft publication.</td>
</tr>
<tr>
<td>0.4</td>
<td>09th August 2013</td>
<td>Updated to include default service implementations</td>
</tr>
</tbody>
</table>
1 Introduction

This RFC will propose some new minor feature enhancements to the existing 1.0 Blueprint Container specification. The feature requests have come from bug reports or through user experience.

2 Application Domain

The following was taken from the Blueprint 1.0 requirements and design documents (RFP 76 & RFC 124, respectively)

The primary domain addressed by this RFP is enterprise Java applications, though a solution to the requirements raised by the RFP should also prove useful in other domains. Examples of such applications include internet web applications providing contact points between the general public and a business or organization (for example, online stores, flight tracking, internet banking etc.), corporate intranet applications (customer-relationship management, inventory etc.), standalone applications (not web-based) such as processing stock feeds and financial data, and “frontoffice” applications (desktop trading etc.). The main focus is on server-side applications.

The enterprise Java marketplace revolves around the Java Platform, Enterprise Edition (formerly known as J2EE) APIs. This includes APIs such as JMS, JPA, EJB, JTA, Java Servlets, JSF, JAX-WS and others. The central component model of JEE is Enterprise JavaBeans (EJBs). In the last few years open source frameworks have become important players in enterprise Java. The Spring Framework is the most widely used component model, and Hibernate the most widely used persistence solution. The combination of Spring and Hibernate is in common use as the basic foundation for building enterprise applications. Other recent developments of note in this space include the EJB 3.0 specification, and the Service Component Architecture project (SCA).

Some core features of the enterprise programming models the market is moving to include:
• A focus on writing business logic in “regular” Java classes that are not required to implement certain APIs or contracts in order to integrate with a container

• Dependency injection: the ability for a component to be “given” its configuration values and references to any collaborators it needs without having to look them up. This keeps the component testable in isolation and reduces environment dependencies. Dependency injection is a special case of Inversion of Control.

• Declarative specification of enterprise services. Transaction and security requirements for example are specified in metadata (typically XML or annotations) keeping the business logic free of such concerns. This also facilitates independent testing of components and reduces environment dependencies.

• Aspects, or aspect-like functionality. The ability to specify in a single place behavior that augments the execution of one or more component operations.

In Spring, components are known as “beans” and the Spring container is responsible for instantiating, configuring, assembling, and decorating bean instances. The Spring container that manages beans is known as an “application context”. Spring supports all of the core features described above.

3 Problem Description

3.1 No Timeout Reference Managers

Blueprint reference managers have mandatory damping, and this damping always waits for a certain amount of time for a service to become available before it throws a ServiceUnavailableException. You can alter the timeout to be shorter than the default, but there is always a wait.

3.2 Blueprint Grace Timeout

When the Blueprint grace period is reached, the Blueprint container no longer checks for dependencies, even if outstanding dependencies are then satisfied. At this point, the only way the container can be refreshed is to restart the bundle. There needs to be a better way to control the life-cycle of blueprint containers that does not risk them becoming zombies.

3.3 Bug 2233 –‘Satisfied’ Lifecycle Notification

The lifecycle events for Blueprint aren't sufficient to get a complete understanding of what Blueprint is currently doing. If blueprint enters the grace period waiting for dependencies, there is no event fired to say when it's dependencies have been satisfied. This is an important piece of information to know when trying to track down blueprint problems.
### 3.4 Injection of Service Properties

If a User wants to access the service properties of an injected service, they must first be injected with the service reference and then programmatically use the service reference to access the properties. Not only is it an inconvenient extra step to have to do to get to the properties, it also ties the bean to the OSGi APIs, which can impact the ability to unit test the bundles.

### 3.5 Bug 2192 - Factory Services Lifecycle issues

If a factory Service is removed and replaced by another factory service, Blueprint doesn't recreate the beans that were created with the original factory.

This is a Blueprint example for the problem:

```xml
<reference id="foo" interface="javax.persistence.EntityManagerFactory"/>

<bean id="not_working_after_update" class="SomeClass">
    <property name="bar">
        <bean factory-component="foo" factory-method="createEntityManager"/>
    </property>
</bean>
```

In the example, the bean created by the service Reference foo, should be deleted and re-created, if the EntityManagerFactory service that foo is bound to, is removed and replaced by another EntityManagerFactory service.

### 3.6 Bug 1295 - Allow Namespace handlers to use inline Blueprint elements

Custom namespace handlers should be able to include / in-line elements from the Blueprint Schema. Bug 1295 was raised against RFC 155 (Namespace Handlers), but requires changes to the core Blueprint specification that are potentially valuable even in the absence of a namespace handler standard.

```xml
<tx:transacted>
    <bean ... />
</tx:transacted>
```

### 3.7 Bug 2406 – Add blueprint extender capability definition

The latest Core OSGi specification introduced the standard capability namespace for extender implementations, and the Blueprint extender needs to implement this new namespace e.g.

```
Provide-Capability: osgi.extender; osgi.extender="osgi.blueprint";
uses="org.osgi.service.blueprint.container,
    org.osgi.service.blueprint.reflect";version:Version="1.1"
```
3.8 Bug 2484 – Address lack of Blueprint opt-in header

Best practice design for extenders is for them to require an opt-in header specified in the extendee bundle's manifest. Blueprint 1.0, however, treats the Bundle-Blueprint header as an opt-out (when no value is specified). This means the default behavior for Blueprint 1.0 is to search all bundles for blueprint configuration, unless they contain the opt-out header, resulting in significant performance issues for large deployments. Blueprint needs a way to address this, ideally without breaking backwards compatibility.

3.9 Allow service properties to be added via <property> elements in Service Manager

When you want to define service properties within a Service Manager, you have to define a service-properties element, and then define entry elements for each property, which seems unnecessarily verbose and over-complicated.

3.10 Make it easier to consume optional services

The blueprint container allows reference-managers to be “optional”, meaning that they may have no backing service. As identified in 3.1, when a reference manager has no backing implementation it can wait for a long time, then it throws ServiceUnavailableException. Whilst reducing this timeout to zero is helpful in error cases for mandatory services, for optional reference managers “no backing service” is a valid, main-path state. Throwing an exception for every one of these invocations is both wasteful and counter-intuitive – it is not an “exceptional” case.

Currently to avoid this scenario blueprint components must be declared as reference-listener objects. They then receive notifications of the reference bind/unbind events. This adds significant additional threading complexity to the blueprint component, and (assuming it is not safe to hold a lock/monitor while calling the reference) there is still a race condition that can result in a ServiceUnavailableException.

Blueprint's purpose is to make exposing and consuming services simpler. The situation outlined above is not simple, and needs to be improved.

3.11 Ensure Blueprint supports new OSGi Service scopes.

RFC 0195 introduces a new Service Factory type, PrototypeServiceFactory, that can return multiple instances of a particular service to the same consuming bundle. Blueprint needs to support this new RFC0195.

3.12 Allow other value types to be used in Service Properties and Map values.

By default, Service Property values and Map Entry values are Strings. If a non-String value type is required, nested entry elements need to be defined, in order to be able to configure the type of the value. It should be easier to specify the type without needing to add the extra nested elements.
4 Requirements

**Blueprint1** – A component MUST be able to request that a Reference Manager should not wait for a service dependency.

**Blueprint2** – It MUST be possible for a Blueprint Container to use the grace period without risking permanent failure (requiring a bundle restart) if the period expires. Once the grace period has been reached, the Blueprint container MAY decide to partially start the beans and services that have had their dependencies satisfied, or it MAY decide to continue to wait for outstanding dependencies. Relevant Lifecycle events MUST be issued for each scenario.

**Blueprint3** – The Blueprint Container MUST issue an additional lifecycle event when the dependencies of a component are satisfied.

**Blueprint4** – A component MUST be able to have the service's properties injected directly without requiring the bean to use OSGi framework APIs.

**Blueprint5** – The Blueprint Container MUST ensure that beans created by factory services are kept in sync with the life-cycle of the factories. For example, if the factory service is replaced, the beans from the old factory should be removed and new beans created using the new factory service.

**Blueprint6** – Custom namespaces MUST be able to include or have inlined Blueprint elements from the Blueprint Schema. This MAY mean changing the Blueprint Schema to define the Blueprint element types upfront, and refer to them throughout the Schema structure, rather than, as it does today, in-lining the element type definitions within the Schema structure.

**Blueprint7** – The Blueprint Container MUST support the extender capability definition.

**Blueprint8** – Blueprint Service Managers MUST support <property> elements.

**Blueprint10** – It MUST be possible to configure Blueprint processing such that the extender does not process any bundles that do not contain the Bundle-Blueprint header.

**Blueprint11** – It MUST be possible to write a blueprint-managed component with an optional reference without having to be registered as a reference-listener, or handling ServiceUnavailableException.

**Blueprint12** – The Blueprint Container MUST support the new Service Scope specification.

**Blueprint13** – It MUST be easier to define non-String values in Service Properties and Maps.
5 Technical Solution

5.1 No Timeout Reference Managers

To allow Reference Managers to be configured to not wait for the services that they reference, the existing optional timeout property will be able to be set to -1. This new value indicates that the reference should not be wait, and should throw a ServiceUnavailableException should the reference not exist at the time the element is processed by the Blueprint container. This attribute is only available on the reference element, and not on the reference-list element.

Reference Manager timeouts can also be specified at the blueprint element level using the new optional timeout attribute. The attribute applies to all Reference Managers for the corresponding Blueprint container. This attribute can have the same values as the timeout attribute on the reference Manager, so setting this to -1 would mean all of the Blueprint container's Reference Managers would not wait their services.

The timeout attribute on individual reference elements will take precedence over the blueprint element's attribute.

Adding these attributes will require a new version of the Blueprint XML schema.

5.2 Blueprint Grace Period enhancements

When the grace period is reached, Blueprint will have two new options available in order to avoid creating 'zombie' Blueprints. These are specified on the existing directive called, blueprint.graceperiod which is specified on the bundle symbolic name. The new values are allowPartial, forever. For clarity, we will also introduce new values fail and none, which will have the same meaning as true and false respectively.

When a value of allowPartial is specified, Blueprint must wait for the graceperiod and when the graceperiod is reached, rather than issuing a FAILURE event and performing “Destroy”, it must create the blueprint container and set up as much of the blueprint as possible, based on the set of satisfied mandatory references in the same way it would if the graceperiod were set to false.

When a value of forever is specified, Blueprint must wait for the graceperiod and when the graceperiod is reached, rather than issuing a FAILURE event and performing “Destroy”, the Blueprint runtime must issue a new GRACE_PERIOD event and begin a new grace period.

Any blueprint bundles that have the new 1.1 namespace will have the allowPartial behaviour as default. If a host bundle has fragment(s) and any of these bundles use the 1.1 namespace, then all these related bundles will use allowPartial by default.

We will need a new version of the Blueprint XML schema for this new attribute to indicate the change in behaviour is required.
5.3 New Satisfied Life-cycleEvent

When the Blueprint Container processes service references and issues a GRACE_PERIOD event because there are missing mandatory dependencies, it will now issue a SATISFIED event when all mandatory dependencies are finally satisfied and before proceeding to the “Register Services” process.

The Blueprint Event property DEPENDENCIES will have the same array of Strings containing the dependencies that was issued in the corresponding GRACE_PERIOD event.

5.4 Injection of Service Properties

When looking for an appropriate bean set method to call for injection of a service object, as a last option, Blueprint will now also look for a method with the following signature:

set{PropertyName}(T ref, java.util.Map<String, ?> props)

PropertyName is the name of the bean property, specified in the property element, for example

```java
public class C {
    public void setProxy(T ref, Map<String, ?> props) {
        ...
    }
}
```

```xml
<reference id="p" interface="T"/>
<bean id="c" class="C">
    <property name="proxy" ref="p"/>
</bean>
```

It is important to note that the injected service properties are an unmodifiable map of properties, and is a snapshot at the time the map is injected.

5.5 Factory Services

Currently when factory services are used, the actual bean created by the factory service, is injected. If the factory service is replaced by another factory service, the blueprint container doesn't replace the existing beans with ones created using this new factory.

The Blueprint container will ensure that whenever a factory service is replaced, all beans created by the original factory service are removed and replaced by new beans created by the new service. The factory service will also now return a damped proxy, rather than the actual bean.

5.6 Inlining of Blueprint elements

In order to support the ability for Custom Namespace Handlers to use inlined Blueprint elements, the Blueprint schema needs to be amended to declare the element types outside of the nested groups and reference them within the groups e.g.

```xml
<xsd:group name="allComponents">
    <xsd:choice>
        <xsd:element ref="service"/>
        <xsd:element ref="ref-list"/>
        <xsd:group ref="targetComponent"/>
    </xsd:choice>
</xsd:group>
```
5.7 Extender Capability

The Core OSGi specification defines a capability namespace for extender implementations. To enable Blueprint extendees to express a requirement for a Blueprint extender and also ensure classpath consistency with the chosen extender, the Blueprint extender must now specify the following capability:

```
Provide-Capability: osgi.extender; osgi.extender="osgi.blueprint";
uses:="org.osgi.service.blueprint.container,
org.osgi.service.blueprint.reflect";version:Version="1.0"
```

The Blueprint container must not extend a bundle that is wired to another provider of the Blueprint extender capability.

A Require-Capability header that wires to this extender capability opts the bundle in to being processed by the blueprint extender. An example of the Require-Capability headers is as follows:

```
Require-Capability: osgi.extender; filter:="(osgi.extender=osgi.blueprint)"; ;
path:List<String>="lib/account.xml, security.bp, cnf/*.xml"
```

The path attribute follows the same pattern as the Bundle-Blueprint header described in section 121.3.4. Unlike, Bundle-Blueprint, absence of a path attribute means the blueprint extender searches for the blueprint xmls in the default location (i.e. OSGI-INF/blueprint/*.xml.

5.8 Allow <property> elements in Service Managers

The current mechanism for defining service properties in Service Managers using the <service-properties> elements and <entry> sub-elements is quite cumbersome, and is designed with the idea that the properties will be put into a Map.

Service Managers should allow one or more <property> elements to be defined, in the same way that Bean Managers do. All defined properties will be used as the service properties when the service is registered.

If a service manager is configured both with the existing <service-properties> mechanism, and also with the <property> elements, these will be merged into a single Map.

If there are any duplicate keys, <property> elements take precedence over <service-properties>.

The Property element will also have the valueType attribute as part of requirement Blueprint13

5.9 Blueprint Extender Configuration (Bug 2484)

Best practice design for extenders is for them to require an opt-in header specified in the extendee bundle's manifest. Blueprint 1.0, however, treats the Bundle-Blueprint header as an opt-out (when no value is specified). This means the default behavior for Blueprint 1.0 is to search all bundles for blueprint configuration, unless they contain the opt-out header, resulting in significant performance issues for large deployments. Rather than change the default blueprint extender behavior this specification adds the ability to configure the extender behavior.
Blueprint 1.0 also has a number of other configurations provided on the Bundle-Blueprint header or in the Blueprint XMLs. These are arguably better suited to being configured on the extender, rather than per-Blueprint. For example, blueprint.timeout, (grace period timeout), default-timeout, default-activation, default-availability.

The Blueprint extender is configured through configuration admin. The pid for the Blueprint extender configuration dictionary is `osgi.blueprint.Extender`.

### 5.9.1 Extendee Header Behavior

The extendee header behavior can be configured by setting the `osgi.blueprint.header` in the `osgi.blueprint.Extender` configuration dictionary. The default value is `OptOut` and configures the extender to behave as defined by the Blueprint 1.0 specification.

The configuration value of `OptIn` switches the default and opt-out behaviors of the extender, specifically:

- Absence of the `Bundle-Blueprint` header means a bundle is not processed for Blueprint configurations.
- If the `Bundle-Blueprint` header is specified with no value then the extender must search for Blueprint configurations in the default location.

The remaining extender processing of the `Bundle-Blueprint` header is unchanged.

### 5.10 Add “Default” implementations for blueprint reference managers

Blueprint reference managers are responsible for locating and tracking suitable services from the OSGi service registry. They are also required to produce a proxy object which wraps the tracked service, as and when the tracked service becomes unavailable a new service replaces it inside the proxy. No reinjection of the proxy object is required.

If the tracked service becomes unavailable and there is no suitable replacement then invocations of the proxy object are required to wait until a replacement is found, or throw `ServiceUnavailableException` if this wait period times out.

This functionality could be usefully extended with the concept of “default service implementations”. Default service implementations can be thought of as locally visible services with the minimum possible ranking. This means that they aren’t visible to other components, will never take precedence over a real service, can be used whenever no suitable service is available.

#### 5.10.1 Declaring a default implementation

A default service implementation can be declared for a reference manager either by using an attribute to refer to an existing blueprint component, or by declaring an inline bean inside a `<default>` element.

**Example 1:**

```xml
<reference interface="java.util.List">
  <default>
    <bean class="java.util.ArrayList"/>
  <default>
</reference>
```
Example 2:

```xml
<bean id="example" class="java.util.ArrayList"/>
<referencedefault="example" interface="java.util.List"/>
```

When a default service implementation is declared in this way it is important that the default object be of the correct type to match the interfaces listed in the reference manager. If this is not the case then the blueprint container implementation is required to throw a `ComponentDefinitionException` when creating the reference manager.

### 5.10.2 Default implementations and the Null Proxy pattern

For some services it may be difficult to provide a suitable default implementation. Equally blueprint managed bundles may not wish to make themselves providers of the service that they are using (for example the `HttpService`), because it will significantly restrict the range of implementations with which they are compatible.

In this case the Null Proxy pattern can be used to support clients that want the benefits of a default implementation, but without the effort of implementing a default. The null proxy pattern involves generating a proxy object that performs no action, and returns `null` or null-like values, for all method calls. This would mean that all void methods do nothing, all methods that return numeric primitives return zero, all boolean methods return false, and all methods that return references return `null`.

The null proxy pattern can be enabled in blueprint using a new Environment Manager "blueprintNullProxy", which has prototype scope. Whenever this environment manager is used the blueprint container must create a new null proxy matching the expected type of the receiver. For example in the following:

```xml
<referencedefault="blueprintNullProxy" interface="java.util.List"/>
```

The type of the blueprintNullProxy created would be `java.util.List`.

### 5.11 Support of new OSGi Service Scopes

RFC0195 introduces the concept of service scopes, and introduces PrototypeServiceFactory, which is a new subtype of ServiceFactory, that can return multiple instances of a particular service object to the same consuming bundle depending on the scopes of the service object and the reference that is consuming the service(s).

The scope of the service will be determined by the scope attribute on the bean, which can have a value of Singleton, Bundle or Prototype. A Scope of Singleton means that the Service Manager will return the same instance of the bean for all references in the Blueprint Container, a bundle scope will return the same instance of the bean for all references in a particular consuming bundle, and a different instance for each bundle, and Prototype scope will return a different instance to all references in all bundles.

The default scope for a service is singleton, as this is the default scope for a bean.

---

The Blueprint container currently notifies beans when a component instance is destroyed via the destroy-method. This is only supported for 'singleton' beans. Prototype scoped bean cleanup is the responsibility of the application. Use of the destroy-method will be extended such that the destroy-method of a bean is also called when a prototype service (a service referencing a prototype bean) releases each bean instance. It should be noted that, as is the case in Blueprint 1.0, the destroy-method will only be called on prototype beans that have been exported as a service. Prototype Beans that are injected into another bean will not have the destroy-method called.

Note: It was discussed on a previous call about having a new unget-method attribute, or whether we should just use the existing destroy-method attribute on the Bean manager, for removal of the bean in non-singleton case.
Services returned from the new PrototypeServiceFactory will have the same lifecycle as any other service, so the Container must track the consumed services and release them when the bundle is stopped. The Blueprint Container must also ensure that all services are registered with a service.scope service-property that indicates the scope of the service. This will be determined by the scope of the bean.

Lookups of beans and services via the Blueprint Container service must also obey the servicebean scopes, so that if a bean injected into a blueprint component, is then looked up via the Blueprint Container Service, the same instance will be returned. The Blueprint Container Service will need to be registered as a ServiceFactory in order to ensure the correct services are returned.

We will also introduce a scope attribute on the Reference Manager that will be used to indicate the scope of services that can be selected. The scope values are singleton, bundle and prototype. If no scope attribute is set, the reference will select services of any scope.

References should not be injected with ServiceObject services.

The scopes of services that the reference manager can reference can also be defined via the filter attribute, using the service.scope service-property.

Blueprint supports the injection of a ServiceReference, a proxy to a service object, or a proxy to a service object along with a Map of the properties under which the service was registered. Blueprint will add public void (ServiceObjects) to the list of supported signatures. Blueprint will continue to call all matching signatures and in no specific order. Note, a user of ServiceObjects much be careful to unget any services obtain through ServiceObjects.getService(…).

5.12 Allow types to be defined in service-properties and Maps

If you want to define a map entry or a service property whose value isn’t a String, then you need to use a nested <value> element in order to be able to set the type, e.g.

<entry key="3">
   <value type="org.osgi.framework.Version">3.14</value>
</entry>

To simplify this process a valueType attribute will be available on the <entry> element so that the above example can be simplified to:

<entry key="3" value="3.14" valueType="org.osgi.framework.Version"/>

Map values can be of any type, but service property values may only be one of the following types:

- **Primitives Number** – int, long, float, double, byte, short, char, boolean
- **Scalar** – String, Integer, Long, Float, Double, Byte, Short, Character, Boolean.
- **Array** – An array of either the allowable primitive or scalar types.
- **Collection** – An object implementing the Collection interface that contains scalar types.

The TservicePropertyEntry and TmapEntry definitions in the blueprint schema will need to be amended for this new attribute.

The valueType attribute will also be defined on the <property> element that requirement Blueprint8 introduces, where service-properties can be defined with property elements.
5.13 Blueprint 1.1 Schema

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--
* $Id: de3c851a280f003f52ccb3d325806f3f084b503f $ *
* Copyright (c) OSGi Alliance (2008, 2009). All Rights Reserved.
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
* http://www.apache.org/licenses/LICENSE-2.0
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
<!-- xxx changed schema version xxx -->
<xsd:schema xmlns="http://www.osgi.org/xmlns/blueprint/v1.1.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.osgi.org/xmlns/blueprint/v1.1.0"
  elementFormDefault="qualified" attributeFormDefault="unqualified"
  version="1.1.0">
  <xsd:annotation>
    <xsd:documentation>
      <!CDATA[
        This is the XML Schema for the OSGi Blueprint service 1.0.0 development descriptor. Blueprint configuration files using this schema must indicate the schema using the blueprint/v1.0.0 namespace. For example,
        <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
          if used as a qualified namespace, "bp" is the recommended namespace prefix.
      ]]>)
    </xsd:documentation>
  </xsd:annotation>

  <!-- Schema elements for core component declarations -->
  <xsd:complexType name="Tcomponent" abstract="true">
    <xsd:annotation>
      <xsd:documentation>
        <![CDATA[
          The Tcomponent type is the base type for top-level Blueprint components. The <bean> <reference>, <service>, and <reference-list> elements are all derived from the Tcomponent type. This type defines an id attribute
        ]]>)
      </xsd:documentation>
    </xsd:annotation>
```
that is used create references between different components. Component elements can also be inlined within other component definitions. The id attribute is not valid when inlined.

```xml
<xsd:attribute name="id" type="xsd:ID" />
<xsd:attribute name="activation" type="Tactivation">
    <xsd:annotation>
        <xsd:documentation><![CDATA[
            The activation attribute for this component. This can either be "eager" or "lazy". If not specified, it defaults to default-activation attribute of the enclosing <blueprint> element.
        ]]></xsd:documentation>
    </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="depends-on" type="TdependsOn">
    <xsd:annotation>
        <![CDATA[
            depends-on identifies (by id) other components that this component depends on. The component only be activated after the depends-on components are successfully activated. Also, if there are <reference> or <reference-list> elements with unstatisfied mandatory references, then the depends-on relationship will also be used to determine whether this service is enabled or not.
        ]]></xsd:documentation>
    </xsd:annotation>
</xsd:attribute>
```

The `<blueprint>` element is the root element for a blueprint configuration file. A blueprint configuration has two sections. The first section (contained within the `<type-converters>` element) identifies components that are used for converting values into different target types. The type converters are optional, so the file does not need to specify a type converter section.

Following the type converters are the component definitions. Components are `<bean>`, `<service>`, `<reference>`, and
<reference-list> elements that identify the bundle components that will be managed by the blueprint service.

<!-- xxx added top level service element xxx -->
<xsd:element name="service" type="Tservice">
    <xsd:annotation>
        <xsd:documentation>
            <![CDATA[
                This <service> element is to allow custom namespaces to inline this element in their own schema definitions.
                This element is not intended for direct use by blueprint bundles.
            ]]>"
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<!-- xxx added top level reference-list element xxx -->
<xsd:element name="reference-list" type="Treference-list">
    <xsd:annotation>
        <xsd:documentation>
            <![CDATA[
                This <reference-list> element is to allow custom namespaces to inline this element in their own schema definitions.
                This element is not intended for direct use by blueprint bundles.
            ]]>"
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<!-- xxx added top level bean element xxx -->
<xsd:element name="bean" type="Tbean">
    <xsd:annotation>
        <xsd:documentation>
            <![CDATA[
                This <bean> element is to allow custom namespaces to inline this element in their own schema definitions.
                This element is not intended for direct use by blueprint bundles.
            ]]>"
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<!-- xxx added top level reference element xxx -->
<xsd:element name="reference" type="Treference">
    <xsd:annotation>
        <xsd:documentation>
This <reference> element is to allow custom namespaces to inline this element in their own schema definitions.

This element is not intended for direct use by blueprint bundles.

<![CDATA[

This <reference> element is to allow custom namespaces to inline this element in their own schema definitions.

This element is not intended for direct use by blueprint bundles.
]]>

</xsd:documentation>
</xsd:annotation>
</xsd:element>

<xsd:complexType name="Tblueprint">
  <xsd:sequence>
    <xsd:element name="description" type="Tdescription"
                 minOccurs="0" />
    <xsd:element name="type-converters" type="Ttype-converters"
                 minOccurs="0" maxOccurs="1" />
  <!-- top-level components -->
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="service" type="Tservice" />
    <xsd:element name="reference-list" type="Treference-list" />
    <xsd:element name="bean" type="Tbean" />
    <xsd:element name="reference" type="Treference" />
    <xsd:any namespace="##other" processContents="strict" />
  </xsd:choice>
  </xsd:sequence>

  <!-- Defaults-->  
  <xsd:attribute name="default-activation" default="eager"
                 type="Tactivation">
    <xsd:annotation>
      <xsd:documentation>
        <![[CDATA[
            Specifies the default activation setting that will be defined for components. If not specified, the global default is "eager".
            Individual components may override the default value.
          ]]]>
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
  <xsd:attribute name="default-timeout" type="Ttimeout"
                 default="300000">
    <xsd:annotation>
      <xsd:documentation>
        <![CDATA[
          Specifies the default timeout value to be used when operations are invoked on unsatisfied service references. If the reference does not change to a satisfied state within the timeout window, an error is raised on the method invocation. The default timeout value is 300000 milliseconds and individual <reference> element can override the specified configuration default. A value of 0 means wait indefinitely, and a value of -1
        ]]]>
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
</xsd:complexType>

Copyright © OSGi Alliance 2014 All Rights Reserved
means do not wait.
]]>
<!-- xxx updated the description above by adding last sentence xxx -->
</xsd:documentation>
</xsd:annotation>
</xsd:attribute>
<xsd:attribute name="default-availability" type="Tavailability"
    default="mandatory">
    <xsd:annotation>
        <xsd:documentation>
            <![CDATA[
                Specifies the default availability value to be used for
                <reference>, and <reference-list> components. The
                normal default is "mandatory", and can be changed by individual
                service reference components.
            ]]>
        </xsd:documentation>
    </xsd:annotation>
</xsd:attribute>
</xsd:complexType>
<xsd:complexType name="Ttype-converters">
    <xsd:annotation>
        <xsd:documentation>
            <![CDATA[
                The type used for the <type-converters> element. The
                <type-converters> section is a set of <bean>, <ref>, or
                <reference> elements that identify the type converter components.
            ]]>
        </xsd:documentation>
    </xsd:annotation>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="bean" type="Tbean" />
        <xsd:element name="reference" type="TReference" />
        <xsd:element name="ref" type="TRef" />
        <xsd:any namespace="##other" processContents="strict" />
    </xsd:choice>
</xsd:complexType>

<!-- Components that provide a reasonable target for injection used for listeners, etc. -->

<xsd:group name="GtargetComponent">
    <xsd:annotation>
        <xsd:documentation>
            <![CDATA[
                A target component is one that can be a target for a
                listener, registration-listener or service elements.
            ]]>
        </xsd:documentation>
    </xsd:annotation>
</xsd:group>
This is used in contexts where the requirement is a single provided object that will implement a particular interface. The provided object is obtained either from a <ref> element or an inlined <bean> or <reference>.
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:choice>
<xsd:element name="bean" type="Tinlined-bean" />
<xsd:element name="reference" type="Tinlined-reference" />
<xsd:element name="ref" type="Tref" />
<xsd:element name="service" type="Tinlined-service" />
<xsd:element name="reference-list" type="Tinlined-reference-list" />
<xsd:group ref="GtargetComponent" />
</xsd:choice>
</xsd:group>
<xsd:group name="GallComponents">
<xsd:annotation>
<xsd:documentation><![CDATA[
An all components is used in contexts where all component element types are values. The set of component elements contains <bean>, <service>, <reference>, <reference-list> and <ref>. ]]>]]>
</xsd:documentation>
</xsd:annotation>
<xsd:choice>
<xsd:element name="service" type="Tinlined-service" />
<xsd:element name="reference-list" type="Tinlined-reference-list" />
<xsd:group ref="GtargetComponent" />
</xsd:choice>
</xsd:group>
<xsd:group name="GbeanElements">
<xsd:annotation>
<xsd:documentation><![CDATA[
A bean elements is a reusable definition of the elements allowed on a <bean> element. ]]>]]>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="description" type="Tdescription" minOccurs="0" />
<xsd:choice minOccurs="0" maxOccurs="unbounded">
<xsd:element name="argument" type="Targument" />
<xsd:element name="property" type="Tproperty" />
<xsd:any namespace="##other" processContents="strict" />
</xsd:choice>
</xsd:sequence>
</xsd:group>
<xsd:complexType name="Tbean">
<xsd:annotation>
<xsd:documentation>
  <![CDATA[
The type definition for a <bean> component. The <bean> attributes provide the characteristics for how to create a bean instance. Constructor arguments and injected properties are specified via child <argument> and <property> elements.
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:complexContent>
  <xsd:extension base="Tcomponent">
    <xsd:group ref="GbeanElements" />
    <xsd:attribute name="class" type="Tclass" />
    <xsd:attribute name="init-method" type="Tmethod" />
    <xsd:attribute name="destroy-method" type="Tmethod" />
    <xsd:attribute name="factory-method" type="Tmethod" />
    <xsd:attribute name="factory-ref" type="Tidref" />
    <xsd:attribute name="scope" type="TbeanScope" />
    <xsd:anyAttribute namespace="##other" processContents="strict" />
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Tinlined-bean">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      The Tinlined-bean type is used for inlined (i.e. non top level) <bean> elements. Those elements have some restrictions on the attributes that can be used to define them.
      ]]>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:restriction base="Tbean">
      <xsd:group ref="GbeanElements" />
      <xsd:attribute name="id" use="prohibited" />
      <xsd:attribute name="depends-on" type="TdependsOn" />
      <xsd:attribute name="activation" use="prohibited" fixed="lazy" />
      <xsd:attribute name="class" type="Tclass" />
      <xsd:attribute name="init-method" type="Tmethod" />
      <xsd:attribute name="destroy-method" use="prohibited" />
      <xsd:attribute name="factory-method" type="Tmethod" />
      <xsd:attribute name="factory-ref" type="Tidref" />
      <xsd:attribute name="scope" use="prohibited" />
      <xsd:anyAttribute namespace="##other" processContents="strict" />
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Targument"
An argument used to create an object defined by a <bean> component. The <argument> elements are the arguments for the bean class constructor or passed to the bean factory method.

The type, if specified, is used to disambiguate the constructor or method signature. Arguments may also be matched up with arguments by explicitly specifying the index position. If the index is used, then all <argument> elements for the bean must also specify the index.

The value and ref attributes are convenience shortcuts to make the <argument> tag easier to code. A fuller set of injected values and types can be specified using one of the "value" type elements.

A property that will be injected into a created <bean> component. The <property> elements correspond to named JavaBean setting methods for a created bean object.

The value and ref attributes are convenience shortcuts to make the <argument> tag easier to code. A fuller set of injected values and types can be specified using one of the "value" type elements.

The valueType attribute allows the type of the attribute to be specified.
<xsd:complexType name="Tkey">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
        The Tkey type defines the element types that are permitted for Map key situations. These can be any of the "value" types other than the <null> element.
      ]]> 
    </xsd:documentation>
  </xsd:annotation>
  <xsd:group ref="GnonnullValue" />
</xsd:complexType>

<xsd:complexType name="Treference">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
        The Treference type defines the <reference> element. These are instances of the Tservicereference type, with the addition of a timeout attribute. If the timeout is not specified, the default-timeout value is inherited from the encapsulating <blueprint> definition. A timeout value of 0 means wait indefinitely, and a value of -1 means do not wait.
      ]]> 
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="Tservicereference">
      <xsd:sequence>
        <!-- xxx added inlined default element -->
        <xsd:element name="default" type="Tinlined-default-bean" minOccurs="0" />
        <xsd:any namespace="#other" minOccurs="0" maxOccurs="unbounded" processContents="strict" />
      </xsd:sequence>
      <xsd:attribute name="timeout" type="Ttimeout" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Tinlined-default-bean">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      ]]> 
    </xsd:documentation>
  </xsd:annotation>
</xsd:complexType>
The Tinlined-default-bean type is used to inline a default bean implementation for a <reference>.

The Tinlined-reference type is used for inlined (i.e. non top level) <reference> elements.

The Reference-list builds in the characteristics of the TserviceReference type to define characteristics of the
<reference-list>. This adds in the characteristics that only apply to collections of references (e.g., member-type).
]]>
</xsd:annotation>
<xsd:complexContent>
<xsd:extension base="TserviceReference">
<xsd:sequence>
  <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
    processContents="strict" />
</xsd:sequence>
<xsd:attribute name="member-type" type="Tservice-use"
  default="service-object" />
</xsd:attribute>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Tinlined-reference-list">
<xsd:annotation>
  <xsd:documentation><![CDATA[
The Tinlined-reference-list type is used for inlined (i.e. non top level)
reference-list elements.
]]></xsd:documentation>
</xsd:annotation>
<xsd:complexContent>
<xsd:restriction base="Treference-list">
<xsd:sequence>
  <xsd:group ref="GserviceReferenceElements" />
  <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
    processContents="strict" />
</xsd:sequence>
<xsd:attribute name="id" use="prohibited" />
<xsd:attribute name="depends-on" type="TdependsOn" />
<xsd:attribute name="activation" use="prohibited"
  fixed="lazy" />
<xsd:attribute name="interface" type="Tclass" />
<xsd:attribute name="filter" type="xsd:normalizedString" />
<xsd:attribute name="component-name" type="Tidref" />
<xsd:attribute name="availability" type="Tavailability" />
<xsd:attribute name="member-type" type="Tservice-use"
  default="service-object" />
  <xsd:anyAttribute namespace="##other"
    processContents="strict" />
</xsd:sequence>
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>

<!-- Reference base class -->
<xsd:complexType name="TserviceReference">
<xsd:annotation>

<![CDATA[
  The TserviceReference type is used for service references.
]]></xsd:documentation>
</xsd:annotation>
<xsd:complexContent>
</xsd:complexContent>
</xsd:complexType>
<xsd:documentation>
  <![CDATA[
  TserviceReference is the base element type used for <reference> and <reference-list> elements. This type defines all of the characteristics common to both sorts of references.
  ]]> 
</xsd:documentation>
</xsd:annotation>
<xsd:complexContent>
  <xsd:extension base="Tcomponent">
    <xsd:sequence>
      <xsd:group ref="GserviceReferenceElements" />
    </xsd:sequence>

    <xsd:attribute name="interface" type="Tclass">
      <xsd:annotation>
        <xsd:documentation>
          <![CDATA[
          The interface that the OSGi service must implement and that will be implemented by the proxy object. This attribute is optional.
          ]]> 
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>

    <xsd:attribute name="filter" type="xsd:normalizedString">
      <xsd:annotation>
        <xsd:documentation>
          <![CDATA[
          A filter string used to narrow the search for a matching service reference.
          ]]> 
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>

    <xsd:attribute name="component-name" type="Tidref">
      <xsd:annotation>
        <xsd:documentation>
          <![CDATA[
          An optional specifier that can be used to match a service definition to one created by a specific blueprint component.
          ]]> 
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>

    <xsd:attribute name="availability" type="Tavailability">
      <xsd:annotation>
        <xsd:documentation>
          <![CDATA[
          Use to control the initial processing of service references at
          ]]> 
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:extension>
</xsd:complexContent>
blueprint context startup. "mandatory" indicates the
context should not start unless the service is available within
the specified context startup period. "optional" indicates
availability of this service is not a requirement at bundle startup.

NOTE: No default is specified because this can be
overridden by the default-availability attribute of the
<blueprint> element.

<!-- xxx added scope attribute xxx -->
<xsd:attribute name="scope" type="TreferenceScope">
<xsd:annotation>
<xsd:documentation>
<![CDATA[
An options specifier that can be used to match to a
service of a specific scope. When not specified the dependency can be
matched to services of any scope. To a reference a "singleton" or "bundle"
scoped service looks the same as the referencing bundle will only ever receive
one instance. Therefore, the scope attribute on a reference only specifies
"bundle" and "prototype" scope, where "bundle" will match both "bundle" and "singleton"
scoped services.
]]>
</xsd:documentation>
</xsd:annotation>
</xsd:attribute>

<!-- xxx added default attribute xxx -->
<xsd:attribute name="default" type="Tidref">
<xsd:annotation>
<xsd:documentation>
<![CDATA[
A default bean implementation to use if the service reference cannot be
bound to a service.
]]>
</xsd:documentation>
</xsd:annotation>
</xsd:attribute>
<xsd:anyAttribute namespace="##other" processContents="strict" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:group name="GserviceReferenceElements">
  <xsd:sequence>
    <xsd:element name="description" type="Tdescription"
      minOccurs="0" />
    <!-- listener -->
    <xsd:element name="reference-listener" type="TReferenceListener"
      minOccurs="0" maxOccurs="unbounded">
      <xsd:annotation>
        <xsd:documentation>
          <![CDATA[
            A definition of a listener that will watch for bind/unbind events
            associated with the service reference. The targetted listener can
            be a <ref> to a <bean> or <reference> element, or an inline
            <bean> or <reference>.
          ]]>}
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
</xsd:group>

<xsd:complexType name="TReferenceListener">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
        TReferenceListener defines a reference listener that is attached
to a <reference> or <reference-list> element. The listener object can be specified as a <ref> or as an inline <bean> or
<reference> component. Listener events are mapped to the indicated
bind or unbind methods.
      ]]>}
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:group ref="GtargetComponent" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="ref" type="Tidref" />
  <xsd:attribute name="bind-method" type="Tmethod" />
  <xsd:attribute name="unbind-method" type="Tmethod" />
</xsd:complexType>

<xsd:simpleType name="Tactivation">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
        Tactivation defines the activation type for components. This is used in this
        schema by the <blueprint> default-activation attribute and the activation attribute.
      ]]>}
    </xsd:documentation>
  </xsd:annotation>
</xsd:simpleType>
<xsd:simpleType name="Tavailability">
  <xsd:annotation>
    <xsd:documentation> <![CDATA[
Tavailability defines an availability attribute type. This is used in this schema by the <blueprint> default-availability attribute and the <reference> and <reference-list> availability attribute.
]]>  
  </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="mandatory" />
    <xsd:enumeration value="optional" />
  </xsd:restriction>
</xsd:simpleType>

<!-- service -->

<xsd:complexType name="Tservice">
  <xsd:annotation>
    <xsd:documentation> <![CDATA[
Tservice is the type for services exported by this blueprint bundle. Services are sourced by either a <ref> to a <bean> component or an <inline> bean component.
]]>  
  </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="Tcomponent">
      <xsd:sequence>
        <xsd:group ref="GserviceElements" />
      </xsd:sequence>
      <xsd:attribute name="interface" type="Tclass">
        <xsd:annotation>
          <xsd:documentation> <![CDATA[
            The interface that this OSGi service will provide.
          ]]>
        </xsd:annotation>
      </xsd:attribute>
      <xsd:attribute name="ref" type="Tidref">
        <xsd:annotation>
          <xsd:documentation> <![CDATA[

Copyright © OSGi Alliance 2014 All Rights Reserved
The ref attribute can be used to specify the component that provides the object exported as an OSGi service. }
</xsd:documentation>
</xsd:annotation>
</xsd:attribute>
<xsd:attribute name="auto-export" type="TautoExportModes"
   default="disabled">
   <xsd:annotation>
   <xsd:documentation>
   <![CDATA[
   If set to a value different from "disabled", the Blueprint Container will introspect the target to discover the set of interfaces or classes that the service will be registered under. ]]> }
   </xsd:documentation>
   </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="ranking" type="xsd:int" default="0">
   <xsd:annotation>
   <xsd:documentation>
   <![CDATA[
   A service ranking value that is added to the service properties the service will be published with. ]]> }
   </xsd:documentation>
   </xsd:annotation>
</xsd:attribute>
<xsd:anyAttribute namespace="##other" processContents="strict" />
<xsd:attribute name="activation" use="prohibited" 
    fixed="lazy" />
<xsd:attribute name="interface" type="Tclass" />
<xsd:attribute name="ref" type="Tidref" />
<xsd:attribute name="auto-export" type="TautoExportModes" 
    default="disabled" />
<xsd:attribute name="ranking" type="xsd:int" default="0" />
<xsd:anyAttribute namespace="##other" 
    processContents="strict" />
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>

<xsd:group name="GbaseServiceElements">
  <xsd:sequence>
    <xsd:element name="description" type="Tdescription" 
        minOccurs="0" />
    <xsd:element name="interfaces" type="Tinterfaces" 
        minOccurs="0"> 
      <xsd:annotation>
        <xsd:documentation>
          <! [CDATA[
            A collection of one or more interface class names this
            service will be registered under. The <service> element also has
            a shortcut interface attribute for the usual case of just
            a single interface being used. This also cannot be used if
            the auto-export attribute is used.
          ]]>}
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="service-properties" type="TserviceProperties" 
        minOccurs="0"> 
      <xsd:annotation>
        <xsd:documentation>
          <![CDATA[
            The service provided when the service is registered. The
            service properties are similar to map elements, but the keys must
            always be strings, and the values are required to be in a narrower
            range.
          ]]>}
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
</xsd:group>

<!-- xxx added <property> -->
<xsd:element name="property" type="TserviceProperty" 
    minOccurs="0" maxOccurs="unbounded"> 
  <xsd:annotation>
    <xsd:documentation>

This can be used in conjunction with <service-properties>.

A set of 0 or more registration listeners attached to this service component. The registration listeners will be notified whenever the service is registered or unregistered from the framework registry.
A registration listener definition. The target registration
listener can be either a <ref> to a <bean> or <service> component, or an
inline <bean> or <service> component definition. The registration-method
and unregistration-method attributes define the methods that will be
called for the respective events.

For the very common case of using a <ref> to a listener component,
the ref attribute may also be used as a shortcut.

<! [CDATA[
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
  <xsd:group ref="GtargetComponent" minOccurs="0" />
</xsd:sequence>
<xsd:attribute name="ref" type="Tidref" />
<xsd:attribute name="registration-method" type="Tmethod" />
<xsd:attribute name="unregistration-method" type="Tmethod" />
</xsd:complexType>

<!-- Values -->
<xsd:group name="Gvalue">
  <xsd:annotation>
    <xsd:documentation>
      <! [CDATA[
The set of "value" types that can be used in any place a value
can be specified. This set includes the <ref> and <idref>
elements, any of the component types (<bean>, <service>, etc.) as inline components, the
generic <value> element for types sourced from string values, any
of the collection types (<set>, <list>, <array>, <map>, <props>), and the
<null> type to inject a null value.
]]>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice>
    <xsd:group ref="GnonnullValue" />
    <xsd:element name="null" type="Tnull" />
  </xsd:choice>
</xsd:group>

<xsd:complexType name="Tnull">
  <xsd:annotation>
    <xsd:documentation>
      <! [CDATA[
The definition for a <null> value type.
]]>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence />
</xsd:complexType>
<xsd:documentation><![CDATA[
The set of "value" types that can be used in any place a non-null value can be specified. This set includes the <ref> and <idref> elements, any of the component types (<bean>, <service>, etc.) as inline components, the generic <value> element for types sourced from string values, and any of the collection types (<set>, <list>, <array>, <map>, <props>).

The <null> type is NOT a member of this group.]]>
</xsd:documentation>
</xsd:complexType>
</xsd:group>

<xsd:complexType name="Tref">
<xsd:annotation>
<xsd:documentation><![CDATA[
Tref is the type used for <ref> elements. This specifies a required component id for the reference component.
]]></xsd:documentation>
</xsd:annotation>
<xsd:attribute name="component-id" type="Tidref" use="required" />
</xsd:complexType>

<xsd:complexType name="Tvalue" mixed="true">
<xsd:annotation>
<xsd:documentation><![CDATA[
Tvalue is the type used for <value> elements. The <value> element is used for types that can be created from a single string value. The string value is the data value for the element. The optional type attribute allows a target conversion value to be explicitly specified.
]]></xsd:documentation>
</xsd:annotation>
</xsd:complexType>
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:attribute name="type" type="Ttype" />
</xsd:complexType>

<!-- Collection Values -->
<xsd:complexType name="TtypedCollection">
<xsd:annotation>
<xsd:documentation>
<![CDATA[
TtypeCollection defines comment attributes shared among different collection types that allow a default value type to be specified.
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:attribute name="value-type" type="Ttype" />
</xsd:complexType>

<xsd:complexType name="Tcollection">
<xsd:annotation>
<xsd:documentation>
<![CDATA[
Tcollection is the base schema type for different ordered collection types. This is shared between the <array>, <list>, and <set> elements.
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:complexContent>
<xsd:extension base="TtypedCollection">
<xsd:group ref="Gvalue" minOccurs="0" maxOccurs="unbounded" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Tprops">
<xsd:annotation>
<xsd:documentation>
<![CDATA[
Tprops is the type used by the <props> value element. The prop elements are pairs of string-valued keys and values.
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="prop" type="Tprop" minOccurs="0" maxOccurs="unbounded" />
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Tprop" mixed="true">

Tprop is a single property element for a <props> value type. The property value can be specified using either the attribute, or as value data for the property element.

Tmap is the base type used for <map> elements. A map may have a default value type specified, so it inherits from the TtypeCollection type. A key type can also be specified, and the map members are created from the entry elements, which require a key/value pair.

TmapEntry is used for <entry> elements nested inside of a <map> element. Each <entry> instance defines a key/value pair that will be added to the Map. Both the keys and values may be arbitrary types. Keys must not be <null> but <null> is permitted for entry values. A default type can be specified for both the keys and the values, but individual keys or values can override the default.
```xml
]>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
  <xsd:element name="key" type="Tkey" minOccurs="0" />
  <xsd:group ref="Gvalue" minOccurs="0" />
</xsd:sequence>
<xsd:attribute name="key" type="TstringValue" />
<xsd:attribute name="key-ref" type="Tidref" />
</xsd:complexType>
<!-- 'service property' element type -->
<xsd:complexType name="TserviceProperties">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      TserviceProperties is used for <service-properties> elements. The syntax is similar to what is defined for <map>, but keys must be string values and there are no type defaults that can be specified. created from the entry elements, which require a key/value pair.
      ]]>>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="entry" type="TservicePropertyEntry" minOccurs="0" maxOccurs="unbounded" />
    <xsd:any namespace="##other" processContents="strict" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<!-- 'entry' element type -->
<xsd:complexType name="TservicePropertyEntry">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      TservicePropertyEntry is an entry value used for the <service-properties> element. This does not allow a child <key> element and there are no key-ref or value-ref attributes.
      ]]>>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:group ref="Gvalue" minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
<!-- xxx added valueType attribute -->
<xsd:attribute name="valueType" type="Ttype" />
</xsd:complexType>
```

<xsd:attribute name="valueType" type="Ttype" />
</xsd:complexType>

<!-- xxx added TserviceProperty type -->
<!-- service 'property' element type -->
<xsd:complexType name="TserviceProperty">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      TserviceProperty is used to specify a <property> on a <service> element. This does not allow a child <key> element and there are no key-ref or value-ref attributes.
      ]]>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:group ref="Gvalue" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="key" type="TstringValue" use="required" />
  <xsd:attribute name="value" type="TstringValue" />
  <xsd:attribute name="valueType" type="Ttype" />
</xsd:complexType>

<!-- General types -->
<xsd:complexType name="Tdescription" mixed="true">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      A generic <description> element type to allow documentation to added to the blueprint configuration.
      ]]>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice minOccurs="0" maxOccurs="unbounded" />
</xsd:complexType>

<xsd:complexType name="Tinterfaces">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      The type definition for the <interfaces> element used for <service> element. This does not allow a child <key> element and there are no key-ref or value-ref attributes.
      ]]>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice minOccurs="1" maxOccurs="unbounded">
    <xsd:element name="value" type="TinterfaceValue" />
  </xsd:choice>
</xsd:complexType>

<xsd:simpleType name="TinterfaceValue">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      A generic <interfaceValue> element type to allow documentation to added to the blueprint configuration.
      ]]>
    </xsd:documentation>
  </xsd:annotation>
</xsd:simpleType>
<xs:simpleType name="Tclass">
  <xs:annotation>
    <xs:documentation><![CDATA[
    Tclass is a base type that should be used for all attributes that refer to java class names.
 ]]></xs:documentation>
  </xs:annotation>
  <xs:restriction base="xsd:NCName"/>
</xs:simpleType>

<xsd:simpleType name="Ttype">
  <xs:annotation>
    <xs:documentation><![CDATA[
    Ttype is a base type that refer to java types such as classes or arrays.
 ]]></xs:documentation>
  </xs:annotation>
  <xs:restriction base="xsd:token">
    <xs:pattern value="[\i-[:][\-[:]*(\[\])*"/>
  </xs:restriction>
</xs:simpleType>

<xsd:simpleType name="Tmethod">
  <xs:annotation>
    <xs:documentation><![CDATA[
    Tmethod is a base type that should be used for all attributes that refer to java method names.
 ]]></xs:documentation>
  </xs:annotation>
  <xs:restriction base="xsd:NCName"/>
</xs:simpleType>

<!-- Should be used for all attributes and elements that refer to method names -->
<xsd:simpleType name="Tidref">
  <xs:annotation>

<! [CDATA[
  TInterfaceValue is used for subelements of the <interfaces> element. This is just a <value>xxxxx</value> element where the contained value is the name of an interface class.
]]> 
</xsd:annotation>
</xsd:restriction>
</xsd:simpleType>
<xsd:documentation>
<![CDATA[
Tidref is a base type that should be used for all attributes that refer to component ids.
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:NCName" />
</xsd:simpleType>

<xsd:simpleType name="TstringValue">
<xsd:annotation>
<xsd:documentation>
<![CDATA[
TstringValue is a base type that should be used for all attributes that refer to raw string values
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:normalizedString" />
</xsd:simpleType>

<xsd:simpleType name="TautoExportModes">
<xsd:annotation>
<xsd:documentation>
<![CDATA[
TautoExportModes is a base type that should be used for export-mode attributes.
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:NMTOKEN">
<xsd:enumeration value="disabled" />
<xsd:enumeration value="interfaces" />
<xsd:enumeration value="class-hierarchy" />
<xsd:enumeration value="all-classes" />
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Ttimeout">
<xsd:annotation>
<xsd:documentation>
<![CDATA[
Ttimeout is a base type that should be used for all attributes that specify timeout values
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:unsignedLong" />
</xsd:simpleType>

<xsd:simpleType name="TdependsOn">
<xsd:annotation>
</xsd:annotation>
</xsd:simpleType>
<xsd:documentation>
  <![CDATA[
  TdependsOn is a base type that should be used for all attributes that specify depends-on relationships
  ]]>
</xsd:documentation>
</xsd:annotation>
<xsd:restriction>
  <xsd:simpleType>
    <xsd:list itemType="Tidref" />
  </xsd:simpleType>
  <xsd:minLength value="1" />
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="TbeanScope">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="singleton" />
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:QName">
        <xsd:pattern value=".*:.*" />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>

<!-- xxx added bundle scope xxx -->
<xsd:annotation>
  <xsd:documentation>
    <![CDATA[
    Copyright © OSGi Alliance 2014 All Rights Reserved
    ]]>
</xsd:documentation>
</xsd:annotation>
<xsd:simpleType name="Tservice-use">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      ]]>
</xsd:documentation>
</xsd:annotation>
<xsd:simpleType name="TreferenceScope">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="bundle" />
        <xsd:enumeration value="prototype" />
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:QName">
        <xsd:pattern value=".*:.*" />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
Indicates the type of object that will be placed within the reference collection. "service-object" indicates the collection contains blueprint proxies for imported services. "service-reference" indicates the collection contains ServiceReference objects matching the target service type.

]]>
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:NMTOKEN">
  <xsd:enumeration value="service-object" />
  <xsd:enumeration value="service-reference" />
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>

6   Considered Alternatives

7   Security Considerations
8 Document Support

8.1 References


8.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Tim Mitchell</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:tim.mitchell@uk.ibm.com">tim.mitchell@uk.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></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>

<table>
<thead>
<tr>
<th>Name</th>
<th>Tim Ward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Paremus</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:tim.ward@paremus.com">tim.ward@paremus.com</a></td>
</tr>
</tbody>
</table>
8.3 Acronyms and Abbreviations

8.4 End of Document
Abstract

The generic capabilities and requirements model in OSGi provides a powerful constraints-based resource selection model, however the API for this is limited to constraints within a single namespace essentially limiting queries to a single type of information. This RFC aims to provide a technical design for the relevant requirements as stated in RFP 148.

This RFC focuses on generic Requirements in the context of the Repository Service API.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

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

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0 Document Information....................................................................................................................... 2
  0.1 License.............................................................................................................................. 2
  0.2 Trademarks..................................................................................................................... 3
  0.3 Feedback....................................................................................................................... 3
  0.4 Table of Contents............................................................................................................. 3
  0.5 Terminology and Document Conventions........................................................................ 4
  0.6 Revision History............................................................................................................. 4

1 Introduction......................................................................................................................... 4

2 Application Domain............................................................................................................. 5

3 Problem Description........................................................................................................... 5

4 Requirements..................................................................................................................... 7

5 Technical Solution............................................................................................................. 7
  5.1 Notes............................................................................................................................. 11

6 Data Transfer Objects......................................................................................................... 11

7 Javadoc............................................................................................................................... 11

8 Considered Alternatives....................................................................................................... 28
0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

0.6 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>09/01/12</td>
<td>David Bosschaert (Red Hat) initial version</td>
</tr>
<tr>
<td>0.2</td>
<td>November, 2012</td>
<td>David Bosschaert (Red Hat) incorporate feedback from the Basel F2F</td>
</tr>
<tr>
<td>0.3</td>
<td>December, 2012</td>
<td>David Bosschaert (Red Hat) feedback from Orlando F2F</td>
</tr>
<tr>
<td>0.4</td>
<td>February, 2013</td>
<td>David Bosschaert (Red Hat) feedback from Austin F2F</td>
</tr>
<tr>
<td>0.5</td>
<td>April, 2013</td>
<td>David Bosschaert (Red Hat) add Javadoc</td>
</tr>
</tbody>
</table>

1 Introduction

The generic capabilities and requirements model in OSGi provides a powerful constraints-based selection model, however the API for selecting resources based on constraints is limited to constraints within a single namespace essentially limiting queries to a single type of information. This RFP 148 describes the uses-cases and requirements that need to be satisfied to allow extending the scope of queries to cover multiple namespaces making it possible to express requirements that cover the widest variety of constraints.

The RFC focuses on generic Requirements in the context of the Repository Service API.
2 Application Domain

A company provides a management solution for deployment of artifacts to a target environment. Supported targets are OSGi frameworks as well as other target containers. The management solution uses OSGi generic capabilities to work with remote artifact repositories which can provide the resources to be provisioned and combines the requirements expressed by the user, transitive requirements defined by the deployables along with the requirements of the target platform. Together these combine into complex expressions that place conditions on a number of criteria. Translated into OSGi Capabilities these constraints translate into requirements on a number of different namespaces. Unfortunately the OSGi Repository API does not provide the expressiveness to specify a requirement on more than one namespace at the time.

For the management agent this means that multiple passes need to be performed, one for each namespace involved, every pass narrowing the resulting set by applying the filter applicable to the current namespace on the result of the previous pass. The end result is that the communication between the management agent and the repositories is not as efficient as it could be, pulling down more resources than needed as only part of the requirements was passed to the Repository for selection.

This RFP relates to the following email conversation on the EEG list:


and this bug in the OSGi bug system:

- https://www.osgi.org/members/bugzilla/show_bug.cgi?id=2208

3 Problem Description

The introduction of the generic capabilities and requirements model in OSGi Core 4.3 has opened up some great possibilities. They can be used in a myriad of different deployment scenarios. In cloud scenarios they can be used to place constraints on the environment, to pull in the bundles needed in order to operate in the current environment or to select qualifying nodes or spawn new nodes. In plain OSGi they can be used to provision dependencies and transitive dependencies. They can be used to ensure an extender is present or to enforce that only bundles that have certain corporate-assigned capabilities can be deployed. Additionally, capabilities can be used by tools to provide certain behavior, for example contract-level imports and by extenders to add behavior, as is done by the ServiceLoader Mediator.

The generic capability and requirement metadata is very useful, however the expressiveness of queries on the metadata is limited to one namespace. Queries on this metadata are expressed as a filter on a certain namespace, for example the following declares that a bundle is needed that export package javax.mail of version between 1.4 and 2.0, which is expressed as a query on the osgi.wiring.package namespace:
Require-Capability: osgi.wiring.package;
   filter:="(& (osgi.wiring.package=javax.mail) (version>=1.4) (!(version>=2)))";

While the LDAP-style querying is quite powerful, it is not possible to express a query that defines a constraint across namespaces. The osgi.identity namespace defines a license attribute, so the following query specifies that a resource should have the EPL license:

Require-Capability: osgi.identity;
   filter:="(license=http://www.opensource.org/licenses/EPL-1.0)"

A common requirement would be for a bundle to provide some capability while at the same time it must have a certain license. While it is possible to combine multiple requirements in a single Require-Capability bundle manifest header, requiring all of them, it is not possible to combine expressions on multiple namespaces in a single Requirement object, when this is expected in the API, for example when communicating with a (possibly remote) Repository. Additionally, multiple requirements expressed in a Require-Capability header do not require to be satisfied by a single resource. The license example clearly shows that both the package requirement as well as the license requirement must be satisfied by the same resource,

The OSGi R5 Requirement interface is defined as follows:

```java
package org.osgi.resource;
public interface Requirement {
   String getNamespace();
   Map<String, String> getDirectives();
   Map<String, Object> getAttributes();
   Resource getResource();
}
```

The OSGi R5 Repository Service interface is defined as:

```java
package org.osgi.service.repository;
public interface Repository {
   Map<Requirement, Collection<Capability>> findProviders(
      Collection<? extends Requirement> requirements);
}
```

Additionally, it is not possible to express an or-relationship across namespaces, neither with the Require-Capability header nor through the Requirement object.

As usage of the generic capabilities and requirements becomes more prevalent making it possible to express complex or composite requirements on capabilities becomes more and more relevant.

In some cases it can already be seen that the namespaces specifications are starting to get 'polluted' by adding attributes such as bundle-symbolic-name and bundle-version to namespaces that really belong on the osgi.wiring.bundle (and osgi.identity) namespaces to other namespaces such as osgi.wiring.package.
4 Requirements

CRR001 – The solution MUST allow a Repository lookup for a combined requirement which spans more than one namespace.

CRR002 – The solution MUST allow combining requirements using the and, or and not logical operators to yield a single complex requirement.

CRR003 – A complex requirement SHOULD itself be a requirement which can be used for further combining; i.e. the solution should follow the Composite pattern.

CRR004 – The solution SHOULD provide a convenient mechanism to construct requirements and complex requirements.

5 Technical Solution

The technical solution proposes to extend the Repository Service API with a findResources method taking a RequirementExpression object. Additionally the repository now provides a builder to make it easy to create Requirements through a RequirementBuilder and RequirementExpression objects via an ExpressionCombiner.

```java
public interface Repository {
    Map<Requirement, Collection<Capability>> findProviders(Collection<? extends Requirement> requirements);
    Collection<Resource> findProviders(RequirementExpression requirementExpression);
    ExpressionCombiner getExpressionCombiner();
    RequirementBuilder newRequirementBuilder(String namespace);
}
```

As a RequirementExpression is not always mappable to a concise set of capabilities (for example, consider 'not' requirements) the findProviders(RequirementExpression re) method returns a collection of matching Resources.

A single RequirementExpression can hold a combination of requirements combined with And, Or or Not operators. A convenience API to construct such requirements is provided through a ExpressionCombiner which can be obtained from the Repository service. Example usage:

```java
Repository repo = ...; // from Service Registry
```
Requirement req1 = repo.newRequirementBuilder("ns1").
    addDirective("filter", "(org.foo.ns1=val1)").
    addDirective("cardinality", "multiple").build();

Map<String, String> dirs = new HashMap<String, String>();
dirs.put("filter", "(org.door.ns2=val2)");
Requirement req2 = repo.newRequirementBuilder("ns2").
    setDirectives(dirs).build();
Requirement req3 = repo.newRequirementBuilder("ns3").build();

ExpressionCombiner expCombiner = repo.getExpressionCombiner();
RequirementExpression re = expCombiner.or(
    expCombiner.not(expCombiner.and(req1, req2)),
    expCombiner.expression(req3));

System.out.println("Using requirement expression:");
printExpression(re);

Collection<Resource> result = _
  repo.findResources(re);

The RequirementBuilder provides a builder-style API to create Requirement objects.

Once all the information is provided to the builder, the desired immutable Requirement object can be obtained by
calling the build() method:

```java
public interface RequirementBuilder {
    // Add an attribute or directive to the Requirement object
    RequirementBuilder addAttribute(String name, Object value);
    RequirementBuilder addDirective(String name, String value);

    // Replace the attributes or directives with the provided map
    RequirementBuilder setAttributes(Map<String, Object> attrs);
    RequirementBuilder setDirectives(Map<String, String> dirs);

    Requirement build();
}
```

The ExpressionCombiner provides a simple API to create RequirementExpression objects. A number of
Requirement or RequirementExpression objects can be combined using and, or and not operators. The result is
always a RequirementExpression object which can be further combined. To facilitate combining a Requirement
and a RequirementExpression together, a Requirement can be converted to a RequirementExpression using the
expression() method.

```java
public interface ExpressionCombiner {
    RequirementExpression and(Requirement... reqs);
    RequirementExpression and(RequirementExpression... reqs);
    RequirementExpression expression(Requirement req);
    RequirementExpression not(Requirement req);
    RequirementExpression not(RequirementExpression req);
    RequirementExpression or(Requirement... reqs);
    RequirementExpression or(RequirementExpression... reqs);
}
```
The RequirementExpression interface hierarchy is as follows:
The interface hierarchy represents a number of holder interfaces that each can hold one or more RequirementExpression objects. The SimpleRequirementExpression can hold and existing org.osgi.resource.Requirement object, allowing it to be represented as a RequirementExpression without the need for modification.

As shown in the example above, RequirementExpression follows the Composite pattern which means that RequirementExpressions themselves can also be used as elements of the operations. It allows nesting.

The RequirementExpression hierarchy is quite simple and can easily be transformed into other representations, making it easy to be sent across the network, given that a Repository is often a remote entity. Note that the interfaces do not implement the java.io.Serializable interface as Java Object Serialization is not expected to be the primary means of remotely serializing these objects. The contents of the Requirement and RequirementExpression objects can be inspected using their APIs and converted in the serialization technology of choice, if needed.

On the receiving side, a RequirementExpression can relatively easily be unwound and processed, the following example shows some code that unwinds a RequirementExpression and creates a string representation of it:

```java
private static String printExpression(RequirementExpression req) {
    if (req instanceof SimpleRequirementExpression) {
        return ((SimpleRequirementExpression) req).getRequirement().toString();
    } else if (req instanceof NotExpression) {
        return "not" + printExpression(((NotExpression) req).getExpression()) + "";
    } else if (req instanceof AndExpression) {
        StringBuilder sb = new StringBuilder();
        AndExpression and = (AndExpression) req;
        List<RequirementExpression> expressions = and.getExpressions();
        for (int i = 0; i < expressions.size(); i++) {
            if (i > 0) sb.append(" and ");
```

Copyright © OSGi Alliance 2013 All Rights Reserved
Given the above example input, the printRequirements outputs the following string:

(not(ns1 and ns2) or ns3)

### 5.1 Notes

The design in this chapter does not address the following cases:

- The technical solution provides an enhancement to the Repository API, however similar issues exist with OSGi Require-Capability bundle manifest headers. The current proposal does not cover this.

### 6 Data Transfer Objects

The OSGi Repository spec 1.0 does not yet define Data Transfer Objects. This addition to the Repository spec does not introduce Data Transfer Objects.

### 7 Javadoc
### OSGi Javadoc

4/19/13 1:11 PM

<table>
<thead>
<tr>
<th>Package Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.service.repository</td>
<td>Repository Service Package Version 1.0.</td>
</tr>
</tbody>
</table>
**Package org.osgi.service.repository**

Repository Service 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>AndExpression</td>
<td>A RequirementExpression representing multiple requirements combined together using the and operator.</td>
<td>14</td>
</tr>
<tr>
<td>ExpressionCombiner</td>
<td>An ExpressionCombiner can be used to combine multiple requirements into a single complex requirement using the and, or and not operators.</td>
<td>17</td>
</tr>
<tr>
<td>NotExpression</td>
<td>A RequirementExpression representing the negative of a requirement, the not operator.</td>
<td>20</td>
</tr>
<tr>
<td>OrExpression</td>
<td>A RequirementExpression representing multiple requirements combined together using the or operator.</td>
<td>21</td>
</tr>
<tr>
<td>Repository</td>
<td>A repository service that contains resources.</td>
<td>22</td>
</tr>
<tr>
<td>RepositoryContent</td>
<td>An accessor for the default content of a resource.</td>
<td>24</td>
</tr>
<tr>
<td>RequirementBuilder</td>
<td>A builder for Requirement objects.</td>
<td>25</td>
</tr>
<tr>
<td>RequirementExpression</td>
<td>The base interface of all Requirement Expressions.</td>
<td>27</td>
</tr>
<tr>
<td>SimpleRequirementExpression</td>
<td>A wrapper to represent a simple org.osgi.resource.Requirement object as a RequirementExpression.</td>
<td>28</td>
</tr>
</tbody>
</table>

### Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentNamespace</td>
<td>Content Capability and Requirement Namespace.</td>
<td>15</td>
</tr>
</tbody>
</table>

**Package org.osgi.service.repository Description**

Repository 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. 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.repository; version="[1.1,2.0)"```

Example import for providers implementing the API in this package:

```import-Package: org.osgi.service.repository; version="[1.1,1.2)"```
Interface AndExpression

org.osgi.service.repository

All Superinterfaces:
  RequirementExpression

public interface AndExpression
  extends RequirementExpression

A RequirementExpression representing multiple requirements combined together using the and operator.

ThreadSafe

Method Summary

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List&lt;RequirementExpression&gt; getRequirements()</td>
<td>14</td>
</tr>
</tbody>
</table>

Obtain the requirements that are combined using the and operator.

Returns: The requirements, represented as RequirementExpression objects.
class ContentNamespace

final public class ContentNamespace extends org.osgi.resource.Namespace

Content Capability and Requirement Namespace.

This class defines the names for the attributes and directives for this namespace. All unspecified capability attributes are of type String and are used as arbitrary matching attributes for the capability. The values associated with the specified directive and attribute keys are of type String, unless otherwise indicated.

Immutable

Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPABILITY_MIME_ATTRIBUTE</td>
<td>static String</td>
<td>The capability attribute that defines the IANA MIME Type/Format for this content.</td>
</tr>
<tr>
<td>CAPABILITY_SIZE_ATTRIBUTE</td>
<td>static String</td>
<td>The capability attribute that contains the size, in bytes, of the content.</td>
</tr>
<tr>
<td>CAPABILITY_URL_ATTRIBUTE</td>
<td>static String</td>
<td>The capability attribute that contains the URL to the content.</td>
</tr>
<tr>
<td>CONTENT_NAMESPACE</td>
<td>static String</td>
<td>Namespace name for content capabilities and requirements.</td>
</tr>
</tbody>
</table>

Fields inherited from class org.osgi.resource.Namespace

CAPABILITY_EFFECTIVE_DIRECTIVE, CAPABILITY_USES_DIRECTIVE, CARDINALITY_MULTIPLE, CARDINALITY_SINGLE, EFFECTIVE_ACTIVE, EFFECTIVE_RESOLVE, REQUIREMENT_CARDINALITY_DIRECTIVE, REQUIREMENT_EFFECTIVE_DIRECTIVE, REQUIREMENT_FILTER_DIRECTIVE, REQUIREMENT_RESOLUTION_DIRECTIVE, RESOLUTION_MANDATORY, RESOLUTION_OPTIONAL

Field Detail

CONTENT_NAMESPACE

public static final String CONTENT_NAMESPACE = "osgi.content"

Namespace name for content capabilities and requirements.

Also, the capability attribute used to specify the unique identifier of the content. This identifier is the SHA-256 hash of the content.

CAPABILITY_URL_ATTRIBUTE

public static final String CAPABILITY_URL_ATTRIBUTE = "url"

The capability attribute that contains the URL to the content.
**CAPABILITY_SIZE_ATTRIBUTE**

```java
public static final String CAPABILITY_SIZE_ATTRIBUTE = "size"
```

The capability attribute that contains the size, in bytes, of the content. The value of this attribute must be of type `Long`.

---

**CAPABILITY_MIME_ATTRIBUTE**

```java
public static final String CAPABILITY_MIME_ATTRIBUTE = "mime"
```

The capability attribute that defines the IANA MIME Type/Format for this content.
public interface ExpressionCombiner

An ExpressionCombiner can be used to combine multiple requirements into a single complex requirement using the and, or and not operators.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>and(org.osgi.resource.Requirement... reqs)</td>
<td>Combine multiple org.osgi.resource.Requirement objects into a single expression using the and operator.</td>
<td>17</td>
</tr>
<tr>
<td>and(RequirementExpression... reqs)</td>
<td>Combine multiple RequirementExpression objects into a single expression using the and operator.</td>
<td>17</td>
</tr>
<tr>
<td>expression(org.osgi.resource.Requirement req)</td>
<td>Convert a org.osgi.resource.Requirement into a RequirementExpression.</td>
<td>18</td>
</tr>
<tr>
<td>not(org.osgi.resource.Requirement req)</td>
<td>Provide the negative of a org.osgi.resource.Requirement.</td>
<td>18</td>
</tr>
<tr>
<td>not(RequirementExpression req)</td>
<td>Provide the negative of a RequirementExpression.</td>
<td>18</td>
</tr>
<tr>
<td>or(org.osgi.resource.Requirement... reqs)</td>
<td>Combine multiple org.osgi.resource.Requirement objects into a single expression using the or operator.</td>
<td>18</td>
</tr>
<tr>
<td>or(RequirementExpression... reqs)</td>
<td>Combine multiple RequirementExpression objects into a single expression using the or operator.</td>
<td>18</td>
</tr>
</tbody>
</table>

### Method Detail

**and**

`and(org.osgi.resource.Requirement... reqs)`

Combine multiple org.osgi.resource.Requirement objects into a single expression using the and operator.

**Parameters:**
- `reqs` - The requirements to combine.

**Returns:**
- A RequirementExpression representing the combined requirements.

**and**

`and(RequirementExpression... reqs)`

Combine multiple RequirementExpression objects into a single expression using the and operator.

**Parameters:**
- `reqs` - The requirements to combine.

**Returns:**
- A RequirementExpression representing the combined requirements.
**expression**

RequirementExpression expression(required osgi.resource.Requirement req)

Convert a org.osgi.resource.Requirement into a RequirementExpression. This can be useful when working with a combination of Requirement and RequirementExpression objects.

**Parameters:**
- req - The requirement to convert.

**Returns:**
- A RequirementExpression representing the requirement.

**not**

RequirementExpression not(required osgi.resource.Requirement req)

Provide the negative of a org.osgi.resource.Requirement.

**Parameters:**
- req - The requirement to provide the negative of.

**Returns:**
- A RequirementExpression representing the negative of the requirement.

**not**

RequirementExpression not(RequirementExpression req)

Provide the negative of a RequirementExpression.

**Parameters:**
- req - The requirement to provide the negative of.

**Returns:**
- A RequirementExpression representing the negative of the requirement.

**or**

RequirementExpression or(Required osgi.resource.Requirement... reqs)

Combine multiple org.osgi.resource.Requirement objects into a single expression using the or operator.

**Parameters:**
- reqs - The requirements to combine.

**Returns:**
- A RequirementExpression representing the combined requirements.

**or**

RequirementExpression or(RequirementExpression... reqs)

Combine multiple RequirementExpression objects into a single expression using the or operator.

**Parameters:**
- reqs - The requirements to combine.
Returns:

A `RequirementExpression` representing the combined requirements.
public interface NotExpression
extends RequirementExpression

A RequirementExpression representing the negative of a requirement, the not operator.

ThreadSafe

### Method Summary

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getRequirement()</td>
<td>20</td>
</tr>
</tbody>
</table>

### Method Detail

**getRequirement**

**RequirementExpression getRequirement()**

Obtain the requirement that is negated.

**Returns:**

The requirement, represented as RequirementExpression.
public interface OrExpression extends RequirementExpression

A RequirementExpression representing multiple requirements combined together using the or operator.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List&lt;RequirementExpression&gt; getRequirements()</td>
<td>21</td>
</tr>
</tbody>
</table>

Obtain the requirements that are combined using the or operator.

Returns: The requirements, represented as RequirementExpression objects.
public interface Repository

A repository service that contains resources.

Repositories may be registered as services and may be used as by a resolve context during resolver operations.

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

ThreadSafe

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String   URL</td>
<td>22</td>
</tr>
<tr>
<td>Service property to provide URLs related to this repository.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>findProviders(Collection&lt;? extends org.osgi.resource.Requirement&gt; requirements)</td>
<td>22</td>
</tr>
<tr>
<td>Find the capabilities that match the specified requirements.</td>
<td></td>
</tr>
<tr>
<td>findProviders(RequirementExpression requirementExpression)</td>
<td>23</td>
</tr>
<tr>
<td>Find the resources that match the specified RequirementExpression.</td>
<td></td>
</tr>
<tr>
<td>getExpressionCombiner()</td>
<td>23</td>
</tr>
<tr>
<td>Obtain an ExpressionCombiner implementation.</td>
<td></td>
</tr>
<tr>
<td>newRequirementBuilder(String namespace)</td>
<td>23</td>
</tr>
<tr>
<td>Obtain a RequirementBuilder implementation which provides a convenient way to create a requirement.</td>
<td></td>
</tr>
</tbody>
</table>

Field Detail

URL

public static final String URL = "repository.url"

Service property to provide URLs related to this repository.

The value of this property must be of type String, String[], or Collection<String>.

Method Detail

findProviders


Find the capabilities that match the specified requirements.
**Interface Repository**

**Parameters:**
requirements - The requirements for which matching capabilities should be returned. Must not be null.

**Returns:**
A map of matching capabilities for the specified requirements. Each specified requirement must appear as a key in the map. If there are no matching capabilities for a specified requirement, then the value in the map for the specified requirement must be an empty collection. The returned map is the property of the caller and can be modified by the caller.

---

**findProviders**

Collection<org.osgi.resource.Resource> findProviders(RequirementExpression requirementExpression)

Find the resources that match the specified RequirementExpression

**Parameters:**
requirementExpression - The RequirementExpression for which matching capabilities should be returned. Must not be null.

**Returns:**
A collection of matching Resources. If there are no matching resources, an empty collection is returned.

---

**getExpressionCombiner**

ExpressionCombiner getExpressionCombiner()

Obtain an ExpressionCombiner implementation. This can be used to combine multiple requirements into a complex requirement using and, or and not operators.

**Returns:**
An ExpressionCombiner.

---

**newRequirementBuilder**

RequirementBuilder newRequirementBuilder(String namespace)

Obtain a RequirementBuilder implementation which provides a convenient way to create a requirement. For example:

```java
Requirement myReq = .newRequirementBuilder("org.foo.ns1")
   .addDirective("filter", "(org.foo.ns1=val1)")
   .addDirective("cardinality", "multiple")
   .build();
```

**Parameters:**
namespace - The namespace for the requirement to be constructed.

**Returns:**
A requirement builder for a requirement in the given namespace.
public interface RepositoryContent

An accessor for the default content of a resource. All org.osgi.resource.Resource objects which represent resources in a Repository must implement this interface. A user of the resource can then cast the org.osgi.resource.Resource object to this type and then obtain an InputStream to the default content of the resource.

ThreadSafe

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputStream getContent()</td>
<td>Returns a new input stream to the default format of this resource.</td>
<td>24</td>
</tr>
</tbody>
</table>

Method Detail

getContent

InputStream getContent()

Returns a new input stream to the default format of this resource.

Returns:
A new input stream for associated resource.
public interface RequirementBuilder

A builder for Requirement objects.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequirementBuilder addAttribute(String name, Object value)</td>
<td>25</td>
</tr>
<tr>
<td>RequirementBuilder addDirective(String name, String value)</td>
<td>25</td>
</tr>
<tr>
<td>build()</td>
<td>26</td>
</tr>
<tr>
<td>RequirementBuilder setAttributes(Map&lt;String, Object&gt; attrs)</td>
<td>25</td>
</tr>
<tr>
<td>RequirementBuilder setDirectives(Map&lt;String, String&gt; dirs)</td>
<td>26</td>
</tr>
<tr>
<td>RequirementBuilder setResource(Resource resource)</td>
<td>26</td>
</tr>
</tbody>
</table>

Method Detail

addAttribute

RequirementBuilder addAttribute(String name, Object value)

Add an attribute to the requirement.

Parameters:

- name - The attribute name.
- value - The attribute value.

Returns:

- a builder object that can be used to further define the requirement.

addDirective

RequirementBuilder addDirective(String name, String value)

Add a directive to the requirement.

Parameters:

- name - The directive name.
- value - The directive value.

Returns:

- a builder object that can be used to further define the requirement.

setAttributes

RequirementBuilder setAttributes(Map<String, Object> attrs)

Set all the attributes to the values in the provided map.

setDirectives

RequirementBuilder setDirectives(Map<String, String> dirs)

Set all the directives to the values in the provided map.

setResource

RequirementBuilder setResource(Resource resource)

Specifies the Resource object for the requirement.
Interface RequirementBuilder

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setAttributes</td>
<td>Set all the attributes to the values in the provided map. This will replace</td>
</tr>
<tr>
<td></td>
<td>any previous attribute set on the builder.</td>
</tr>
<tr>
<td>Parameters: attrs</td>
<td>- The map of attributes to use.</td>
</tr>
<tr>
<td>Returns: a builder object</td>
<td>that can be used to further define the requirement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setDirectives</td>
<td>Set all the directives to the values in the provided map. This will replace</td>
</tr>
<tr>
<td></td>
<td>any previous directives set on the builder.</td>
</tr>
<tr>
<td>Parameters: dirs</td>
<td>- The map of directives to use.</td>
</tr>
<tr>
<td>Returns: a builder object</td>
<td>that can be used to further define the requirement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setResource</td>
<td>Specifies the Resource object for the requirement. Note that providing a</td>
</tr>
<tr>
<td></td>
<td>resource is optional.</td>
</tr>
<tr>
<td>Parameters: resource</td>
<td>- The resource for the requirement. Will overwrite any previous resource if</td>
</tr>
<tr>
<td></td>
<td>provided.</td>
</tr>
<tr>
<td>Returns: a builder object</td>
<td>that can be used to further define the requirement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>Build the requirement according to the specification provided to the builder.</td>
</tr>
<tr>
<td>Returns: the requirement</td>
<td></td>
</tr>
</tbody>
</table>
**Interface RequirementExpression**

org.osgi.service.repository

**All Known Subinterfaces:**
- AndExpression, NotExpression, OrExpression, SimpleRequirementExpression

**public interface RequirementExpression**

The base interface of all Requirement Expressions. Requirement Expression objects will always be of one of its sub-interfaces.

**ThreadSafe**
Interface SimpleRequirementExpression

```
org.osgi.service.repository
```

**All Superinterfaces:**

```
RequirementExpression
```

```
public interface SimpleRequirementExpression
extends RequirementExpression
```

A wrapper to represent a simple `org.osgi.resource.Requirement` object as a `RequirementExpression`.

**ThreadSafe**

---

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>org.osgi.resource.Requirement</code> getRequirement()</td>
<td>28</td>
</tr>
</tbody>
</table>

```
Obtain the wrapped org.osgi.resource.Requirement object.
```

### Method Detail

#### getRequirement

```
org.osgi.resource.Requirement getRequirement()
```

```
Obtain the wrapped org.osgi.resource.Requirement object.
```

**Returns:**

```
The wrapped object.
```

Java API documentation generated with DocFlex/Doclet v1.5.6

DocFlex/Doclet is both a multi-format Javadoc doclet and a free edition of DocFlex/Javadoc. If you need to customize your Javadoc without writing a full-blown doclet from scratch, DocFlex/Javadoc may be the only tool able to help you! Find out more at www.docflex.com

---

### 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.
9 Security Considerations

No change from the OSGi Repository 1.0 specification.

10 Document Support

10.1 References


10.2 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></td>
</tr>
<tr>
<td>Voice</td>
<td></td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:david@redhat.com">david@redhat.com</a></td>
</tr>
</tbody>
</table>

10.3 Acronyms and Abbreviations

10.4 End of Document
The current Http Service specification is based on Servlet API 2.1. As such it misses newer functionality such as Servlet Filters or event listeners. In addition use of the service does not support the recent whiteboard pattern approach. This RFC lists requirement to create a new Http Whiteboard Service specification.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

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

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

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable, worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise
I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0 Document Information .................................................................................................................. 2
  0.1 License ................................................................................................................................. 2
  0.2 Trademarks ........................................................................................................................ 3
  0.3 Feedback ............................................................................................................................ 3
  0.4 Table of Contents ............................................................................................................... 3
  0.5 Terminology and Document Conventions ...................................................................... 4
  0.6 Revision History ................................................................................................................. 4

1 Introduction .................................................................................................................................. 8

2 Application Domain ......................................................................................................................... 8

3 Problem Description ......................................................................................................................... 9
  3.1 Support for dated Servlet API 2.1 ...................................................................................... 9
  3.2 Dependency on the HttpService service .......................................................................... 9
  3.3 Configuration ....................................................................................................................... 9

4 Requirements .................................................................................................................................. 10
  4.1 New Http Whiteboard Service API .................................................................................. 10

5 Technical Solution ......................................................................................................................... 11
  5.1 New Http Whiteboard Service API .................................................................................. 11
    5.1.1 Servlet API Reference Version ................................................................................ 11
    5.1.2 Annotations ............................................................................................................ 12
    5.1.3 Web Application Events ......................................................................................... 12
    5.1.4 Relationship to Servlet Container ........................................................................ 12
    5.1.5 Http Runtime Service ........................................................................................... 15
5.1.6 Servlet API Exports

5.2 Whiteboard Registration Support

5.2.1 Target HttpService

5.2.2 ServletContextHelper for servlets, servlet filters, resources, and listeners

5.2.3 Lifecycle of servlets, servlet filters, resources, and listeners

5.2.4 Servlet Registration

5.2.5 Servlet Filter Registration

5.2.6 Resources

5.2.7 Event Listeners

5.2.8 Error Pages

5.3 Provided Capability

5.3.1 osgi.whiteboard Namespace

5.4 Potential Update of the Http Service

5.4.1 Supported Servlet API

5.4.2 Relationship to the Http Runtime Service

5.4.3 Coexistence of the Http Service and the Http Whiteboard Service

6 Data Transfer Objects

7 Javadoc

8 Considered Alternatives

8.1 New methods to register Servlets and Filters

8.2 Web Application Events

8.2.1 Limiting events

8.2.2 Event Admin Service

8.3 HTTP Sessions

8.4 Resources

8.5 Deprecated HttpService

9 Security Considerations

10 Document Support

10.1 References

10.2 Author’s Address

10.3 Acronyms and Abbreviations

10.4 End of Document

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

0.6 Revision History

The last named individual in this history is currently responsible for this document.
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Initial  | 02.11.12 | Initial Version  
Felix Meschberger, Adobe Systems Incorporated, fmeschbe@adobe.com |
| Update   | 27.01.12 | Update on Feedback from Orlando F2F and BJ Hargrave on the CPEG mailing list.  
Felix Meschberger, Adobe Systems Incorporated, fmeschbe@adobe.com |
| Update   | 28.01.12 | Update on feedback from Austin F2F  
• Removal of new registration/unregistration methods  
• Clarification of Servlet API 3 registration methods  
• Definition of the osgi.whiteboard namespace  
• Minor clarifications and fixes  
Felix Meschberger, Adobe Systems Incorporated, fmeschbe@adobe.com |
| Update   | 16.04.13 | Update with feedback from Cologne F2F  
• Annotations and asynchronous processing  
Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 22.05.13 | Added section about listener registration  
Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 15.07.13 | Updated with feedback from Palo Alto F2F  
• Updated listener handling  
• Clarified service lifecycle handling  
• Renamed “pattern” property to “path”  
Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 29.07.13 | Updated with feedback from CPEG call  
• Changed handling of multiple whiteboard implementation  
Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Update   | 15.08.13   | Updated with feedback from BJ (partially already mentioned at the Palo Alto F2F):  
• Clean up requirements list  
• Several clarifications / rewordings, samples  
• Moved DTOs to org.osgi.dto.service.http  
• Added security permissions  
Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 23.08.13   | Update with feedback from CPEG call and add missing pieces:  
• use different registration properties for servlets and servlet filters  
• add notes about service life cycle and clarify properties for each service  
• Use consistent naming, changed the flow of chapters for easier reading  
Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 01.10.13   | Update with feedback from CPEG call:  
• Reformat by moving common properties into separate chapter  
• Use prototype scope  
Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 25.10.13   | Update with bug 2468 (RFC 180)  
Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 2013-11-11 | API/Javadoc improvements  
BJ Hargrave, IBM |
| Update   | 28.02.14   | Update with feedback from Austin F2F  
– new abstract class as a replacement for HttpContext  
– add dispatching configuration for servlet filters  
– clarify mapping of ServletContext methods  
– allow a path configuration for contexts  
– added serviceld property to DTOs  
– Renamed ResourceServletDTO to ResourceDTO (bug 2572)  
– Created DTO hierarchy, context as the root (bug 2572)  
Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Update   | 03.04.14 | - Update with feedback from CPEG call  
|          |          | - Undeprecate HttpService and move properties from runtime to service registration properties  
|          |          | - Remove shared attribute from ServletContextHelper  
|          |          | - Clarify session handling  
|          |          | - Minor clarifications  
|          |          | Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 28.04.14 | - Deprecate HttpService (again) and move service registration properties to HttpServiceRuntime.  
|          |          | Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 07.05.14 | Update with feedback from Basel F2F  
|          |          | - Leave HttpService as is  
|          |          | - Update DTOs to contain failed bindings  
|          |          | - Rename specification to Http Whiteboard Service  
|          |          | - Add chapter on potential updates to the Http Service  
|          |          | - Move DTOs to separate package with new “root” DTO: Runtime DTO  
|          |          | - Add new RequestInfoDTO  
|          |          | Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 22.05.14 | Add regular expression support for servlet filters and change filter ordering (highest service ranking is first now)  
|          |          | Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 05.06.14 | Fix minor typos  
|          |          | David Bosschaert, Adobe Systems Incorporated, bosschae@adobe.com |
| Update   | 02.07.14 | Clarify handling of ServletContextHelper/ServletContext (#2695)  
|          |          | Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 22.07.14 | Update of servlet filter handling and several ServletContextHelper  
|          |          | Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| Update   | 25.07.14 | Make path property required of ServletContextHelper  
|          |          | Add a way to override the default ServletContextHelper  
|          |          | Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
### 1 Introduction

The OSGi Specifications currently only contain limited specification support for creating Web Applications in an OSGi context:

- **Http Service Specification** based on Servlet API 2.1. Apart from being based an old Servlet API version and being silent about how more recent versions are supported the main problem with this specification is that a provider of servlets and resources has to grab the Http Service first before being able to register servlets and resources. There is no whiteboard pattern support.

- **Web Applications Specification** basically just defines how existing web applications may be enhanced with OSGi Manifest headers and deployed into the OSGi Framework as-is. This is fine for moving existing web applications with minimal changes into the OSGi framework.

Some thoughts are already listed on the OSGi Community Wiki at [http://wiki.osgi.org/wiki/WebExperience](http://wiki.osgi.org/wiki/WebExperience).

### 2 Application Domain

Developers need to use the full extend of current Servlet API specifications (as of this writing Servlet API 3.0 is the most recent version). As such there is a need to register servlet filters and event listeners.
3 Problem Description

3.1 Support for dated Servlet API 2.1
Current support for web applications using the Http Service in traditional OSGi based applications is limited to servlets and resources. From the current Servlet API 3.0 specification the following functionality is missing:

- Servlet Filters
- Servlet Event Listeners
- Asynchronous Requests

At this moment some of this missing functionality is covered in a proprietary way. Examples are the Apache Felix Http Whiteboard support or the OPS4J Pax Web collection of bundles.

3.2 Dependency on the HttpService service
Currently the HttpService service (or one of them if multiple services exist in a framework) must be accessed to be able to register servlets and/or resources. In addition to register a servlet or resource an instance of the HttpContext interface is required.

This makes it very cumbersome to easily register servlets and resources. Particularly it is hard to come up with an HttpContext instance which for example uses an authentication mechanism available in the framework to implement the handleSecurity method.

To reduce (or simplify) this dependency it would be helpful to just register servlets as services and have them registered with a matching Http Service in a whiteboard pattern style. Likewise registration of static resources would be supported in an extender pattern style.

At this moment some of this missing functionality is covered in a proprietary way. Examples are the Apache Felix Http Whiteboard support or the OPS4J Pax Web collection of bundles.

3.3 Configuration
The Http Service specification currently declares a number of framework properties to configure the Http Service. This raises a number of issues:

- Unable to dynamically reconfigure the Http Service in an easy way
- Incomplete configuration. For example the local interface to bind to is not an official configuration property
- When the Http Service is implemented as bridge to a Servlet Container in which the OSGi framework is deployed (e.g. as part of a Web Application) these properties have no effect.

In addition the actual configuration of an Http Service instance cannot be easily queried/introspected.
4 Requirements

4.1 New Http Whiteboard Service API

HS-1 The solution MUST provide a Http Whiteboard Service specification to refer to Servlet API 3.0 specification and define to what extent the Http Whiteboard Service provides support.

HS-2 The solution MUST provide a service API to support Servlet registration with patterns as defined by the Servlet API specification (Section 12.2, Specification of Mappings, in the Servlet API 3.0 specification). This requirement aligns servlet registration to functionality provided by the Servlet API web application descriptor (web.xml).

HS-3 The solution MUST provide a service API to support registration of Servlet API filters with patterns as defined by the Servlet API specification (Section 12.2, Specification of Mappings, in the Servlet API 3.0 specification) or referring to servlets by their names. This requirement aligns mapping filters to requests to functionality provided by the Servlet API web application descriptor (web.xml).

HS-4 The solution MUST add support for error page configuration.

HS-5 The solution MUST define how registered servlets and servlet filters are named.

HS-6 The solution MUST clarify ServletContext implementation for both standalone and bridged Http Whiteboard Service implementations.

HS-7 The solution MUST clarify the ServletContext scope of Servlet API listeners registered through the Http Whiteboard Service.

HS-8 The solution MUST define runtime attribute of the Http Whiteboard Service to reflect configuration of the service.

HS-9 The solution MUST define whiteboard registration of servlet services with the Http Whiteboard Service.

HS-10 The solution MUST define whiteboard registration of filter services with the Http Whiteboard Service.

HS-11 The solution MUST define whiteboard registration of servlet listener services with the Http Whiteboard Service.

HS-12 The solution MUST define registration of OSGi HttpContext services used for Servlet and Filter registration.

HS-13 The solution MUST define how servlets, filters, and servlet listener services are matched with Http Whiteboard Service implementations.

HS-14 The solution MUST define whiteboard registration of static resources.

HS-15 The solution MUST define whiteboard registration of error pages.

HS-16 The solution MUST define a capability for the whiteboard pattern registration in one of the standard namespaces (or a new namespace to be defined in the Chapter 135, Common Namespaces...
5 Technical Solution

The Http Whiteboard Specification consists of three parts:

- Whiteboard Registration support for servlets, servlet filters, listeners, resources and servlet context.
- Http Runtime Service to introspect the current state of the whiteboard service
- Potential Updates to the Http Service specification

The Http Whiteboard Specification provides all the functionality currently covered by the Http Service Specification Version 1.3 with the difference of using the whiteboard pattern instead of a programmatic API.

5.1 New Http Whiteboard Service API

The goal of this specification is to make the registration of more elements of the Web Application Descriptor available to OSGi applications compared to the current Http Service:

- Servlets may be registered with more than one pattern (instead of a single alias)
- Servlet filters (introduced in Servlet API 2.3)
- Error pages (introduced in Servlet API 2.2)
- Event Listener (introduced in Servlet API 2.3)

Of the remaining elements defined in the Web Application descriptors, MIME type mapping and login configuration will be provided through a similar concept as the HttpContext interface of the Http Service specification.

Resources (EJB) are not supported by the Http Whiteboard Service because these are outside of the scope of the Http Whiteboard Service and are supported by other mechanisms in the OSGi framework such as the service registry or through JNDI.

5.1.1 Servlet API Reference Version

Implementations of the Http Whiteboard Service Specification 1.0 are based on the Servlet API Specification Version 3.0. The implementation may support a higher version than Version 3.0 and must declare this through those methods as well. The actual version supported is exposed through the ServletContext.getMajorVersion() and ServletContext.getMinorVersion() methods.
5.1.2 Annotations

Annotations defined in the Servlet API Specifications must be ignored by an implementation of the Http Whiteboard Service Specification. This is to avoid class path scanning and rather going the OSGi way. In addition this avoids unwanted situations where servlets are registered just by the fact that a specific class is contained in a bundle – this could lead to the servlet registered twice, with the wrong context or registered at all.

Implementations of the Http Whiteboard Service Specification may support annotations through an additional proprietary opt-in mechanism like a manifest header or require capability.

5.1.3 Web Application Events

Starting with Servlet API 2.3 event listener interfaces have been defined to be notified of various events during the web application and request processing life cycle. The Http Whiteboard Service supports all listeners as defined in section 11.2, Event Listeners, of the Servlet API 3.0 specification [3].

5.1.4 Relationship to Servlet Container

Implementations of the Http Whiteboard Service specification will generally be backed by actual implementations of the Servlet API specification such as Apache Tomcat or Jetty. There also exist implementations which bridge into a servlet container into which the OSGi Framework has been deployed as a web application, for example the Apache Felix Http Service Bridge or the Equinox Http Service Bridge.

As such an Http Whiteboard Service implementation will live in a servlet context and all servlets, servlet filters, listeners and resources registered through the Http Whiteboard Service will be backed by the same ServletContext. However as explained in the next section, based on the configuration servlets, servlet filters, listeners and resources might get different ServletContext objects which delegate certain functionality to the backing context. In the case of a bridged usage the relationship looks like below where ServletContext A is the backing context.

Servlet Container 1:n
  Webapp 1:1
    ServletContext[A] 1:1
      Http Service 1:n
        HttpServletContextHelper 1:1
        ServletContext[B]

With respect to Web Applications two areas need clarification as to how they are segregated or shared amongst the servlets, servlet filters, listeners and resources:

- ServletContext objects used for servlet and servlet filter initialization
- Http Sessions acquired by servlets and servlet filters through the HttpServletRequest

5.1.4.1 HttpContext, ServletContextHelper and ServletContext

The Http Service specification currently defines the correlation between an HttpContext used for Servlet (and now Filter) registration and the ServletContext used for the Servlet and Filter initialization as follows:

Servlet objects require a ServletContext object. This object provides a number of functions to access the Http Service Java Servlet environment. It is created by the implementation of the Http Service for each unique
HttpContext object with which a Servlet object is registered. Thus, Servlet objects registered with the same.HttpContext object must also share the same.ServletContext object.

The Servlet API 3.0 contains functionality which would require an extension of the existing HttpContext interface. As enhancing this interface would require a major version change on the Http Service specification and would break existing implementations of the Http Service and to make this specification independent from the HttpService specification this specification introduces a new abstract class ServletContextHelper. Own implementations of this class must inherit from the abstract class and register themselves as ServletContextHelper services.

A ServletContextHelper is also handling resource processing and is therefore usually associated with a bundle to serve resources from a bundle. However, as a ServletContextHelper can be associated with services from different bundles, a ServletContextHelper should be implemented as a ServiceFactory ensuring services from different bundles use a different ServletContextHelper object referencing the bundle of the whiteboarded service. The default ServletContextHelper must be implemented in this way.

Servlets, resources, servlet filters, and listeners are registered with a ServletContextHelper. A ServletContext object is created by the implementation of the Http Whiteboard Service for each ServletContextHelper service with which a whiteboard service is registered. Thus, whiteboard services registered with the same ServletContextHelper object must also share the same ServletContext, even if the ServletContextHelper is registered as a ServiceFactory.

However, the ServletContext instance passed to a whiteboard service may be different for services from different bundles as a proxy is passed on. The proxy handles dispatching resource calls to the correct ServletContextHelper instance and maybe also other methods like the getClassLoader() method (see below table).

The table lists all methods of the ServletContext interface and how these methods should be implemented:

<table>
<thead>
<tr>
<th>Method</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>getClassLoader (Servlet API &gt;= 3.0)</td>
<td>This method must return the class loader of the whiteboard service. An implementation of the Http Whiteboard Service can achieve this by returning separate instances of the ServletContext to each whiteboard service. Such an instance would be a facade of the used Servlet Context but has access to the context of the bundle of the whiteboard service.</td>
</tr>
<tr>
<td>get(ContextPath (Servlet API &gt;= 2.5)</td>
<td>Backed by Servlet Container and might return ServletContextHelper specific path. See 5.2.2</td>
</tr>
<tr>
<td>get(Context(String)</td>
<td>Backed by Servlet Container. Always returns the backing context</td>
</tr>
<tr>
<td>getMajorVersion()</td>
<td>Backed by Servlet Container</td>
</tr>
<tr>
<td>getMinorVersion()</td>
<td>Backed by Servlet Container</td>
</tr>
<tr>
<td>getMimeType(String)</td>
<td>Backed by ServletContextHelper</td>
</tr>
<tr>
<td>getEffectiveMinorVersion()</td>
<td>Same as getMinorVersion()</td>
</tr>
<tr>
<td>getEffectiveMajorVersion()</td>
<td>Same as getMajorVersion()</td>
</tr>
<tr>
<td>getResourcePaths(String)</td>
<td>Backed by ServletContextHelper</td>
</tr>
<tr>
<td>getResource(String)</td>
<td>Backed by ServletContextHelper</td>
</tr>
<tr>
<td>getResourceAsStream()</td>
<td>Backed by ServletContextHelper</td>
</tr>
</tbody>
</table>
getRequestDispatcher(String)  See note 1.
getNamedDispatcher(String)  See note 1.
getServlet(String)  Backed by Servlet Container
getServlets()  Backed by Servlet Container
getServletNames()  Backed by Servlet Container
log(String)  Backed by Servlet Container
log(Exception, String)  Backed by Servlet Container
log(String, Throwable)  Backed by Servlet Container
getRealPath(String)  Backed by ServletContextHelper
getServerInfo()  Backed by Servlet Container
getInitParameter(String)  See note 2.
getInitParameterNames()  See note 2.
getAttribute(String)  Managed per ServletContextHelper
getAttributeNames()  Managed per ServletContextHelper
setAttribute(String, Object)  Managed per ServletContextHelper
removeAttribute(String)  Managed per ServletContextHelper
getServletContextName()  See note 3.

Programmatic Web Application configuration methods  See note 4.

Notes:

1. If the argument matches a servlet registered by the Http Whiteboard Service this method must be handled by the Http Service. Otherwise it must be backed by the Servlet Container.

2. In addition to the underlying ServletContext's initialization parameters, the Http Whiteboard Service exposes its own service registration properties and runtime attributes as ServletContext initialization parameters.

3. By default this method is backed by the Servlet Container. If the ServletContextHelper has a name, this name is returned.

4. These methods for programmatic registration of servlets, servlet filters, and listeners in a Servlet API 3 servlet container should throw IllegalStateException.

5.1.4.2 Http Sessions

HTTP Sessions are defined by chapter 7, Sessions, in the Servlet API 3.0 [3]. specification. HTTP Sessions are managed by the servlet container separately for each web application with the session ID sent back and forth between client and server as a cookie or as a request parameter. Assuming the session ID cookie, this is attached to the servlet context path.

Session handling is usually done by the servlet container outside of the Http Whiteboard Service implementation. Therefore the container manages a single session for the Http Whiteboard Service implementation. The Http
Whiteboard Service implementation must make sure to create a wrapper session object for each ServletContextHelper which manages the session attributes as a separate set for each ServletContextHelper.

5.1.4.3 Lifecycle of Request Handling Objects

When the Http Whiteboard Service receives a request it establishes the processing pipeline based on the available services (filters, servlets, and listeners) at this point of time and executes this pipeline. Between establishing the pipeline and finishing the processing, services used in this pipeline might become unregistered. It is up to the implementation of such a service whether it throws a servlet exception if it gets executed in that case or not. (This is basically the same as with the current Http Service and a servlet gets unregistered while it is processing a request).

5.1.4.4 Asynchronous Requests

If the implementation supports Servlet API 3.0 (or higher), servlets might use the asynchronous request handling feature. However as the servlet might not be available when the processing continues a servlet exception will be thrown.

A servlet or filter supporting the asynchronous mode must declare this with the appropriate service property osgi.http.whiteboard.servlet.asyncSupported or osgi.http.whiteboard.filter.asyncSupported.

5.1.5 Http Runtime Service

The Http Runtime Service provides introspection the current state of the Http Whiteboard Service with respect to used whiteboard services and failed usages of the whiteboard services.

5.1.5.1 Runtime Attributes

The Http Runtime Service implementation must define a set of runtime attributes which can be used by whiteboard services to associate themselves with a specific implementation. This is done via the osgi.http.whiteboard.target service property. The runtime attributes can be examined as service properties of the HttpServiceRuntime service registration. The runtime attributes should include the following attribute.

| osgi.http.endpoint | A String+ value of Http Whiteboard Service endpoints provided as URLs e.g. http://192.168.1.10:8080/ or relative paths, e.g. /myapp/. Relative paths may be used if the scheme and authority parts of the URLs are not known such as in a bridged Http Whiteboard Service implementation. If the Http Whiteboard Service is serving the root context and neither scheme nor authority is known, the value of the property is "/". Each entry must end with a slash. |

5.1.5.2 Configuration

The level of configurability of the Http Whiteboard Service may vary between implementations. Some implementations may allow to configure down to the interface and port level (for example the Jetty based Apache Felix implementation) while others don't allow anything to be configured (for example a bridging implementation where configuration is done in the servlet container).
If an implementation supports configuration, such configuration should be supplied via the Configuration Admin Service.


This draft explicitly does not define a standard configuration PID for the Http Whiteboard Service implementation to be used as this would prevent scalability/usual implementation patterns, like using factory configurations or having multiple Http Whiteboard Service implementations at runtime.

### 5.1.5.3 Diagnostics

See chapter 6, Data Transfer Objects, on the diagnostic API.

### 5.1.6 Servlet API Exports

The Http Whiteboard Service implementation bundle is not required to export the Servlet API Java Packages. If it does so, the bundle must obey semantic versioning and support the portable Java Contracts as defined in RFC 180 [4]. The following sections list the entry for providing the contract for Servlet API 3.0 and Servlet API 2.5.

If the Servlet API is provided by another bundle, the Http Whiteboard Service implementation is a consumer of that API and should require the contract. The bundle providing the Servlet API should provide the corresponding contract.

#### 5.1.6.1 Providing Servlet API 3.0

```plaintext
Export-Package: javax.servlet; javax.servlet.http, javax.servlet.annotation, javax.servlet.descriptor; version=2.6
Provide-Capability: osgi.contract; osgi.contract=JavaServlet;
    version:List<Version>="2.5,3.0";
    uses:="javax.servlet, javax.servlet.http, javax.servlet.annotation, javax.servlet.descriptor"
```

Providing Servlet API 2.5

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

### 5.2 Whiteboard Registration Support

With whiteboard registration support for servlets, listeners, resources, servlet filters, and ServletContextHelper services it is easy to register these web application elements without tracking the any service. The information required for the registration is provided with service registration properties.

The following table lists the common properties for whiteboard registration of servlets, listeners, resources and servlet filters. They are explained in more detailed in the next chapters.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>osgi.http.whiteboard.context.select</td>
<td>String</td>
<td>The value of this service property refers to a ServletContextHelper service. If this property is missing, the default context is used. If the property does start with a ( it is used as a filter expression against the service properties of the</td>
</tr>
</tbody>
</table>
ServletContextHelper, otherwise it is matched against the name of the ServletContextHelper. The special value "*" selects all available ServletContextHelper services. If a value is specified and no matching context exists, the whiteboard service is ignored. This situation is reflected in the failed DTOs. If more than one service matches, the whiteboard service is associated with each matching ServletContextHelper.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>osgi.http.whiteboard.target</td>
<td>String</td>
<td>The value of this service property is an LDAP filter expression which selects the Http Whiteboard Service implementation to process the whiteboard service.</td>
</tr>
</tbody>
</table>

### 5.2.1 Target HttpService

Servlet, servlet filter, listener, and resource services may register with a `osgi.http.whiteboard.target` property containing a filter expression. A Http Whiteboard Service about to process a servlet, servlet filter, listener, or resource must match that filter against its runtime attributes. Only if the filter matches, the servlet, servlet filter, listener, or resource is used by the Http Whiteboard Service. For example a whiteboard service registered with the property

```
 osgi.http.whiteboard.target = "(osgi.http.implementation.name=Admin)"
```

**must only be used by an Http Whiteboard Service with the runtime attribute osgi.http.implementation.name having the value admin.**

Without such a target property all available Http Whiteboard Services are matching. Even if a target property is used, still several Http Whiteboard Services might match. However, a servlet, listener, resource, or servlet filter service must only be used by a single Http Whiteboard Service. To prevent multiple uses a whiteboard support implementation must ensure to process such objects only with a single Http Whiteboard Service by themselves. If more than a single whiteboard support implementation is active at runtime, there is the potential that a servlet, listener, resource or servlet filter is used by more than a single Http Whiteboard Service. In this case such objects should use the target property described above making sure that not more than one Http Whiteboard Service matches the filter expression.

If more than one Http Whiteboard Service is matching and the servlet, servlet filter, resource and listener services are registered with prototype scope (see RFC 195 Service Scopes), this service will be used by all matching Http Whiteboard Services. If more than one Http Whiteboard Service is matching and the servlet, servlet filter, resource and listener services are registered with bundle scope, the service will be used by all matching Http Whiteboard Services registered by different bundles but only with one Http Whiteboard Service from the same bundle.

If more than one Http Whiteboard Service match, e.g. in the absence of the `osgi.http.whiteboard.target` property, any one Http Whiteboard Service may use the service. It is undefined which Http Service this is.

The runtime attributes of the Http Whiteboard Service using the servlet, servlet filter, listener, or resource service are exposed as ServletContext initialization parameters.

### 5.2.2 ServletContextHelper for servlets, servlet filters, resources, and listeners

By default the whiteboard support is associating servlets, servlet filters, listeners, and resources with the default ServletContextHelper of the targeted Http Whiteboard Service. Additional ServletContextHelper services can be made available through the whiteboard support. In this case the ServletContextHelper service must specify the `osgi.http.whiteboard.context.name` service property. This name can be referenced by a servlet, servlet filter, listener, or resource services. The default ServletContextHelper has the predefined name `default`. If a
context is registered with this name, it is overriding the default context. Such a context can be registered with any valid path and is not required to be the root.

If there are multiple, usable ServletContextHelper services registered with the same context name, the Http Whiteboard Service implementation must use the ServletContextHelper with the highest service ranking. This might lead to re-binding the servlet, servlet filter, listener or resource e.g. if a new usable ServletContextHelper with a higher service ranking arrives or the current used ServletContextHelper is unregistered (see section 5.2.3).

If a servlet or servlet filter is used by an Http Whiteboard Service implementation, the implementation calls the init() method of the servlet or servlet filter which gets a configuration object (ServletConfig or FilterContext) that returns a ServletContext object. The Http Whiteboard Service implementation is creating a ServletContext object for each ServletContextHelper it is using. Therefore servlets and servlet filters used by the same Http Whiteboard Service and referencing the same ServletContextHelper, share the ServletContext object.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>osgi.http.whiteboard.context.name</td>
<td>String+</td>
<td>For ServletContextHelper services this property is required and identifies the service when referred to by a whiteboard service. ServletContextHelper services without this property are ignored. The name must follow the symbolic name definition. If the name is invalid, the context is not used and this situation is reflected in the failure DTOs.</td>
</tr>
<tr>
<td>osgi.http.whiteboard.context.path</td>
<td>String</td>
<td>Required property for defining a context path for the context. The value is either a slash for the root or it must start with a slash but not end with a slash. Valid characters are as defined in rfc3986#section-3.3. If the value is invalid, the context is not used and this situation is reflected in the failure DTOs.</td>
</tr>
</tbody>
</table>

A ServletContextHelper is registered with a context path, like in the example below is the default context and two custom contexts registered with different paths.

```
Http Service 1:n
    ServletContextHelper [name=default, path=/]
    ServletContextHelper [name=A, path=/app-a]
    ServletContextHelper [name=B, path=/app-b]
```

Assuming the root of the Http Whiteboard Service is accessible via the path /myserver, servlets registered with the above default context helper will be registered under /myserver, servlets registered with helper A will be registered under /myserver/app-a and servlets registered with helper B will be registered under /myserver/app-b. Different ServletContextHelper services with different names may use the same context path property. If there are several contexts for the same path, the one with the highest service ranking is used.

If a servlet context helper is registered with several names, the first one in the list is considered the official name and the other are aliases. The method getServletContextName will return the first name.

When a request is received, the ServletContextHelper with the longest matching path is used for processing the request. The method handleSecurity(final HttpServletRequest request, final HttpServletResponse response) from the ServletContextHelper object is called before any request listener, filter or servlet is called. If the call to this method returns false, no further processing must take place.
The execution pipeline consisting of request listeners, filters and the servlet (see section 5.1.4.3) is assembled of the servlet matching the request and those listeners and filters which match the request. Only services associated with the processing ServletContextHelper are taken into account. Listeners and filters are chained based on their service ranking, highest ranking first.

The using bundle for a ServletContextHelper must be the bundle of the whiteboard service – this is in order to correctly support ServiceFactory registrations for ServletContextHelper services.

### 5.2.3 Lifecycle of servlets, servlet filters, resources, and listeners

If a servlet, servlet filter, resource or listener service is used by an Http Whiteboard Service implementation, the following order of actions are performed:

1. The service is get from the service registry
2. For servlets and servlet filters, init() is called

If the service is not used anymore, these actions are performed:

1. For servlets and servlet filters, destroy() is called
2. The service is released

As servlet and servlet filter services might come and go as well as ServletContextHelper services might come and go, the whiteboard service registration can be very dynamic. Therefore servlet and servlet filter services might transition between used by a Http Whiteboard Service implementation to not being used and back to be used. As in this case, init() and destroy() are called each time the service is used, the recommended way to register servlet and servlet filter services is to use the prototype scope. In that case a new instance is created for each usage. If the prototype scope is not used, the service should be implemented in a reentrant way and be prepared that after a call of destroy() a new initialization through init() might follow.

If a servlet or servlet filter is not registered with the prototype scope, it can only be associated with a single ServletContextHelper to conform to the Servlet API specification. The service will be used with the first matching ServletContextHelper with a higher service ranking. In this case, a rebinding to the context with the higher ranking occurs. The failure DTOs capture the situation of matching servlet context helpers which are not associated to the service. Once such a service is not associated with a ServletContextHelper anymore, it is free to be used with another context in the future.

### 5.2.4 Servlet Registration

Servlets are registered with a list of patterns in the osgi.http.whiteboard.servlet.pattern service registration property. These patterns are defined by the Servlet API 3.0 specification [3], in section 12.2, Specification of Mappings:

- A string beginning with a '/' character and ending with a '/\*\*' suffix is used for path mapping.
- A string beginning with a '*.' prefix is used as an extension mapping.
- The empty string ('"') is a special URL pattern that exactly maps to the application's context root, i.e., requests of the form http://host:port/<context- root>/\. In this case the path info is '/' and the servlet path and context path is empty string ('"').
- A string containing only the '/' character indicates the "default" servlet of the application. In this case the servlet path is the request URI minus the context path and the path info is null.
All other strings are used for exact matches only.

A servlet may register itself with the property `osgi.http.whiteboard.servlet.name` which can be used by servlet filters to address this servlet. If the servlet does not set this property, the servlet name defaults to the fully qualified class name of the service object. If there is more than one servlet with the same name and also associated with the same ServletContextHelper, then the servlet with the highest service ranking is used and the other servlet is ignored. The same happens if there is more than a single servlet using the exact value for a pattern within the same ServletContextHelper.

If a servlet is used by an Http Whiteboard Service implementation, the init() method of the servlet will be called. Once the servlet is no longer be used by the Http Whiteboard Service implementation the destroy() method will be called. All service registration properties starting with `servlet.init.*` are passed as servlet init parameters to the servlet as well as all runtime attributes of the Http Runtime Service. The service registration properties have precedence over the runtime attributes.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>osgi.http.whiteboard.servlet.name</code></td>
<td>String</td>
<td>The name of a servlet. This name is used as the value of the <code>ServletConfig.getServletName()</code> method and defaults to the fully qualified name of the service object's class.</td>
</tr>
<tr>
<td><code>osgi.http.whiteboard.servlet.pattern</code></td>
<td>String+</td>
<td>Registration patterns for the servlet.</td>
</tr>
<tr>
<td><code>osgi.http.whiteboard.servlet.asyncSupported</code></td>
<td>Boolean</td>
<td>Declares whether the servlet supports asynchronous operation mode.</td>
</tr>
<tr>
<td><code>osgi.http.whiteboard.servlet.errorPage</code></td>
<td>String+</td>
<td>Register the servlet as an error page for error code and/or exception; the value may be fully qualified exception type or three digit HTTP status code. Any value not being a three digit number is assumed to be a fully qualified class name.</td>
</tr>
<tr>
<td><code>servlet.init.*</code></td>
<td>String+</td>
<td>Properties starting with this prefix are passed as servlet init parameters to the init method of the servlet.</td>
</tr>
</tbody>
</table>

5.2.5 Servlet Filter Registration

Servlet filters have been introduced into the Servlet API specification in Version 2.3 and thus far support for them has been absent in the Http Service specification. This specification adds support to register servlet filters through the whiteboard pattern. A servlet filter can be registered with path patterns like a servlet or a servlet filter may be mapped to a specific servlet by referencing the servlet's name.

A servlet filter can set the `osgi.http.whiteboard.filter.pattern` property to path patterns as defined by the Servlet API 3.0 specification [3], in section 12.2, Specification of Mappings or a servlet filter can set the `osgi.http.whiteboard.filter.regex` property to regular expressions matched against the path. A servlet filter can also reference servlets by name using the `osgi.http.whiteboard.filter.servlet` property. A servlet filter matches the request if at least one of the provided properties matches.

A servlet filter may register itself with the property `osgi.http.whiteboard.filter.name`. If the servlet filter does not set this property, the servlet filter name defaults to the fully qualified class name of the service object. If there is more than one servlet filter with the same name and also associated with the same ServletContextHelper, then the servlet filter with the highest service ranking is used and the other servlet filter is ignored.
The servlet filter dispatcher configuration can be set with the property
osgi.http.whiteboard.filter.dispatcher. Allowed string values are REQUEST, ASYNC, ERROR, INCLUDE, and FORWARD. The default for a filter is REQUEST. See Java servlet specification 3.0, Chapter 6.2.5 for more information.

If a servlet filter is used by an Http Whiteboard Service implementation, the init() method of the servlet filter will be called. Once the servlet filter is no longer be used by the Http Whiteboard Service implementation, the destroy() method will be called. All service registration properties starting with filter.init. are passed as init parameters to the filter as well as all runtime attributes of the Http Runtime Service. The service registration properties have precedence over the runtime attributes.

Depending on the service configuration through the osgi.http.whiteboard.context.select service property, the servlet filter might be bound to one or more ServletContextHelper services. A servlet filter is only run as part of a request targeting a servlet selecting the same ServletContextHelper.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>osgi.http.whiteboard.filter.name</td>
<td>String</td>
<td>The name of a servlet filter. This name is used as the value of the FilterConfig.getFilterName() method and defaults to the fully qualified name of the service object's class.</td>
</tr>
<tr>
<td>osgi.http.whiteboard.filter.pattern</td>
<td>String+</td>
<td>Registration property for a servlet filter to apply this filter to the url paths.</td>
</tr>
<tr>
<td>osgi.http.whiteboard.filter.servlet</td>
<td>String+</td>
<td>Registration property for a servlet filter to apply this filter to the referenced servlet.</td>
</tr>
<tr>
<td>osgi.http.whiteboard.filter.regex</td>
<td>String+</td>
<td>Registration property for a servlet filter to apply this filter to the url paths. The values must be regular expressions following the Java syntax defined in java.util.regex.Pattern.</td>
</tr>
<tr>
<td>osgi.http.whiteboard.filter.asyncSupported</td>
<td>Boolean</td>
<td>Declares whether the servlet filter supports asynchronous operation mode.</td>
</tr>
<tr>
<td>osgi.http.whiteboard.filter.dispatcher</td>
<td>String+</td>
<td>Registration property for a servlet filter to set the associated dispatcher configuration when the filter should be called.</td>
</tr>
<tr>
<td>filter.init.*</td>
<td>String+</td>
<td>Properties starting with this prefix are passed as filter init parameters to the init method of the filter.</td>
</tr>
</tbody>
</table>

### 5.2.6 Resources

To register resources through the whiteboard an instance of the type javax.servlet.Servlet is registered as a regular servlet with the additional osgi.http.whiteboard.resource.prefix servlet registration property. The osgi.http.whiteboard.servlet.pattern property must also be specified.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>osgi.http.whiteboard.resource.prefix</td>
<td>String</td>
<td>This prefix is used to map a requested resource to the bundle's entries.</td>
</tr>
</tbody>
</table>

Example using DS:

```java
@Component(property={"osgi.http.whiteboard.context.name=resource-context"]) public class ResourceHttpContext extends ServletContextHelper{
    ...
```
@Component(service = javax.servlet.Servlet.class, scope=ServiceScope.PROTOTYPE, property={
    "osgi.http.whiteboard.servlet.pattern=/files/**",
    "osgi.http.whiteboard.resource.prefix=/tmp",
    "osgi.http.whiteboard.context.select=resource-context"})
public class MyResource extends HttpServlet {
    ...
}

5.2.7 Event Listeners

Event listeners register themselves under the interface(s) they are implementing. This specification supports:

- ServletContextListener
- ServletContextAttributeListener
- ServletRequestListener
- ServletRequestAttributeListener
- HttpSessionListener
- HttpSessionAttributeListener

Events are sent to all listeners registered in the OSGi service registry based on their registration properties. Each listener is associated with a ServletContextHelper as described in section 5.2.2.

The Http Whiteboard Service implementation gets the listeners from the service registry as soon as the associated ServletContextHelper is established and releases them when the ServletContextHelper is not available any more or the listener is unregistered.

5.2.7.1 ServletContextListener and ServletContextAttributeListener

The ServletContextListener receives events after the Http Whiteboard Service implementation has started and the corresponding ServletContextHelper is available and when either the ServletContextHelper becomes unavailable or the Http Whiteboard Service implementation is about to stop. A newly registered listener will be called with the contextInitialized method either if the ServletContextHelper is available or when the ServletContextHelper becomes available. As soon as the ServletContextHelper or the Http Whiteboard Service implementation becomes unavailable, the contextDestroyed method is called. The Http Service implementation holds the listener as long as the ServletContextHelper is available. ServletContextAttributeListeners are held for the same period of time.

Due to the nature of using the whiteboard pattern, a ServletContextListener or ServletContextAttributeListener might be registered after other services like servlets or filters for the same ServletContextHelper. And the listeners might also disappear before other services are unregistered. In this case the ordering of events might not be the same as in a typical web application where the listeners are always called first. If other whiteboard services require a listener to be setup before this whiteboard service is initialized, it needs to create some kind of dependency on the listener to ensure the correct ordering. For example, if Declarative Services is used to implement the whiteboard services, a servlet could have a mandatory reference to the listener service.
A Http Whiteboard Servlet implementation must process registered ServletContextListener and ServletContextAttributeListeners before any other whiteboard service.

Methods in the ServletContext object handed to the contextInitialized method of a registered ServletContextListener to programmatically register servlets, servlet filters, and listeners are not supported and should throw UnsupportedOperationException. The particular reason for not supporting these methods is the mismatch between the lifecycle of the servlet container and the lifecycle of the bundle trying to programmatically register Servlets, Filters, or Listeners.

If implementations of the Http Whiteboard Service decide to support dynamic registration through the servlet context from within the contextInitialized method, they should require a proprietary opt-in mechanism like a manifest header or require capability.

5.2.7.2 Supported Non-Whiteboard Listeners

The servlet specification defines some listener interfaces where the listener is not registered through the web.xml or the corresponding api. For example, the HttpSessionActivationListener is supported for objects registered as session attributes. For these types of listeners, whiteboard registration is neither required nor supported. Implementation of this specification support the following listeners:

- HttpSessionActivationListener
- HttpSessionBindingListener
- AsyncListener

5.2.8 Error Pages

A servlet can be marked to be called in case of errors, either if an exception is thrown during request processing or if a servlet uses the sendError method with a status code of 4xx or 5xx.

The service property osgi.http.whiteboard.servlet.errorPage can be specified on a servlet service. The property values can be an HTTP status code or the fully qualified name of an exception. If such a status code is set via sendError or such an exception is thrown, this servlet is invoked to render an error page. A servlet serving error page requests does not need to set the osgi.http.whiteboard.servlet.pattern service property. If it does so, the servlet can be called by using the path, but might wish to do so to serve regular requests as well.

Example:

```java
@Component(service = javax.servlet.Servlet.class, scope=ServiceScope.PROTOTYPE,
    property={
        "osgi.http.whiteboard.servlet.errorPage=java.io.IOException",
        "osgi.http.whiteboard.servlet.errorPage=500"})
public class MyErrorServlet extends HttpServlet {
    ...
}
```

The above servlet is invoked if the status code 500 is sent via sendError or if an IOException occurs. In general error pages are invoked according to the rules defined in section 10.9.2 in the servlet specification.

If there is more than one error page registered for the same exception or error code within a single ServletContextHelper, the one with the highest service ranking is used.
5.3 Provided Capability

The Http Whiteboard Service implementation bundle must provide the osgi.whiteboard capability for "osgi.http". For example:

```
Provide-Capability: osgi.whiteboard;
    osgi.whiteboard="osgi.http";
    uses:"javax.servlet, javax.servlet.http";
    version:Version="1.3"
```

The Http Whiteboard Service implementation must provide support for all whiteboard service types as outlined in this specification.

5.3.1 osgi.whiteboard Namespace

The whiteboard pattern leverages the OSGi service registry as a registry for objects. In the context of Http Whiteboard Service, servlets can be registered as services and the Http Whiteboard Service implementation uses these services to interact with the servlets.

A Whiteboard Services Consumer is a bundle that monitors the life cycle events of specific services to use their functionality when the specific services are active. It can use metadata (service properties) to control its functionality. Whiteboard Services Providers, register such services, therefore have a dependency on the Whiteboard Services Consumer that can be modeled with the osgi.whiteboard namespace. The definition for this namespace can be found in the following table and the WhiteboardNamespace class.

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>M/O</th>
<th>Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>osgi.whiteboard</td>
<td>CA</td>
<td>M</td>
<td>String</td>
<td>symbolic-name</td>
<td>A symbolic name for the whiteboard services consumer. These names are defined in their respective specifications and should in general use the specification top level package name. For example, org.acme.foo. The OSGi Alliance reserves names that start with osgi.</td>
</tr>
<tr>
<td>version</td>
<td>CA</td>
<td>M</td>
<td>Version</td>
<td>version</td>
<td>A version. This version must correspond to the specification of the whiteboard services consumer.</td>
</tr>
</tbody>
</table>

Specifications for whiteboard services consumers (Http Whiteboard Service, Event Admin, etc.) should specify the values for these attributes. Whiteboard services consumers that provide such a capability should list the packages that they use in their implementation in the uses directive of that capability to ensure class space consistency. Whiteboard services consumers can consume a whiteboard services provider even if that bundle does not require the whiteboard consumer unless the specification explicitly forbids this. For example an Http Whiteboard Service could declare its capability with the following manifest header:

```
Provide-Capability: osgi.whiteboard;
    osgi.whiteboard="osgi.http";
    uses:"javax.servlet, javax.servlet.http";
    version:Version="1.3"
```

A bundle that depends on an Http Whiteboard Service implementation could require such a whiteboard consumer with the following manifest header:

```
Require-Capability: osgi.whiteboard;
    filter="(&(osgi.whiteboard=osgi.http)(version>=1.3)(&(version>=2.0)))"
```
5.4 Potential Update of the Http Service

In order to get support for Servlet API 3.0 and support of the runtime inspection, the Http Service specification could be updated at least with the following chapters.

5.4.1 Supported Servlet API

The Http Service implementation can support declaring the supported servlet API as explained in section 5.1.6.

5.4.2 Relationship to the Http Runtime Service

If an Http Service implementation wants to support introspection through the Http Service Runtime service, the Http Service Runtime service must have a service registration property osgi.http.runtime.httpservice.serviceid containing the service id of the Http Service service.

As the DTOs contain a field with the service id of the whiteboarded services, in the case of servlets or resources registered via the Http Service, the Http Runtime Service needs to generate unique negative ids for these instances.

5.4.3 Coexistence of the Http Service and the Http Whiteboard Service

If an implementation implements both, the Http Service and the Http Whiteboard Service, services registered by any of those means coexist potentially within the same Http context. In the case of a clash between a service registered through the whiteboard pattern and the Http Service, the whiteboard service wins as it has a higher service ranking.

The default Http Context from the Http Service represents the same servlet context as the default ServletContextHelper from the Http Whiteboard service.

6 Data Transfer Objects

This RFC defines an API to retrieve administrative information from the Http Whiteboard Service implementation. The HttpServiceRuntime service is introduced and can be called to obtain various DTOs.

The DTOs for the various services contain the field serviceld. In the case of whiteboard services this value is the value of the service.id property of the corresponding service registration. If the Http Runtime Service supports introspection of an Http Service, for servlets and resources directly registered through the HttpService API, the Http Runtime Service implementation assigns each registration a unique negative service id starting with -1 and decreasing for each registration.

In the case of a clash, e.g. two servlets registered with the same path on the same servlet context, only the service with the highest service id is used. The service(s) with the lower service id(s) are unused. The Http Service Runtime provides DTOs for those unused services as well as failures when using a service like an exception thrown by a servlet from within the init() method in order to find setup problems.
See the JavaDoc for details.

7 Javadoc
<table>
<thead>
<tr>
<th>Package Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.service.http.context</td>
<td>Http Service Context Package Version 1.0.</td>
</tr>
<tr>
<td>org.osgi.service.http.runtime</td>
<td>Http Service Runtime Package Version 1.0.</td>
</tr>
<tr>
<td>org.osgi.service.http.runtime.dto</td>
<td>Http Service Runtime DTO Package Version 1.0.</td>
</tr>
<tr>
<td>org.osgi.service.http.whiteboard</td>
<td>Http Service Whiteboard Package Version 1.0.</td>
</tr>
</tbody>
</table>
Package org.osgi.service.http.context

@org.osgi.annotation.versioning.Version(value="1.0")

Http Service Context Package Version 1.0.

See:
  Description

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServletContext Helper</td>
<td>29</td>
</tr>
<tr>
<td>Helper service for the servlet context used by whiteboard services for HTTP requests.</td>
<td></td>
</tr>
</tbody>
</table>

Package org.osgi.service.http.context Description

Http Service Context 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.http.context; version="[1.0,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.service.http.context; version="[1.0,1.1)"
Helper service for the servlet context used by whiteboard services for HTTP requests.

This service defines methods that the Http Whiteboard Service implementation may call to get information for a request when dealing with whiteboard services.

Servlets, servlet filters, resources, and listeners services may be associated with an ServletContextHelper service. Those whiteboard services that are associated using the same ServletContextHelper object will share the same ServletContext object.

If no ServletContextHelper service is associated, a default ServletContextHelper is used. The behavior of the methods on the default ServletContextHelper is defined as follows:

1. `getMimeType` - Does not define any customized MIME types for the Content-Type header in the response, and calls the parent context if it exists. Otherwise it always returns null.
2. `handleSecurity` - Performs implementation-defined authentication on the request.
3. `getResource` - Assumes the named resource is in the bundle of the whiteboard service. This method calls the whiteboard service bundle's Bundle.getEntry method, and returns the appropriate URL to access the resource. On a Java runtime environment that supports permissions, the Http Whiteboard Service needs to be granted `org.osgi.framework.AdminPermission[*,RESOURCE]`.
4. `getResourcePaths` - Assumes that the resources are in the bundle of the whiteboard service. This method calls Bundle.findEntries method, and returns the found entries. On a Java runtime environment that supports permissions, the Http Whiteboard Service needs to be granted `org.osgi.framework.AdminPermission[*,RESOURCE]`.
5. `getRealPath` - This method returns null.

A context can be registered with a service property to define a path under which all services registered with this context are reachable.

See Also: `HttpWhiteboardConstants.HTTP_WHITEBOARD_CONTEXT_NAME`, `HttpWhiteboardConstants.HTTP_WHITEBOARD_CONTEXT_PATH`

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static String</td>
<td></td>
</tr>
<tr>
<td><code>AUTHENTICATION_TYPE</code></td>
<td>30</td>
</tr>
<tr>
<td>HttpServletRequest</td>
<td>attribute specifying the scheme used in authentication.</td>
</tr>
<tr>
<td>static String</td>
<td></td>
</tr>
<tr>
<td><code>AUTHORIZATION</code></td>
<td>30</td>
</tr>
<tr>
<td>HttpServletRequest</td>
<td>attribute specifying the Authorization object obtained from the org.osgi.service.useradmin.UserAdmin service.</td>
</tr>
<tr>
<td>static String</td>
<td></td>
</tr>
<tr>
<td><code>REMOTE_USER</code></td>
<td>30</td>
</tr>
<tr>
<td>HttpServletRequest</td>
<td>attribute specifying the name of the authenticated user.</td>
</tr>
</tbody>
</table>

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ServletContextHelper()</code></td>
<td>30</td>
</tr>
<tr>
<td>Default constructor</td>
<td></td>
</tr>
</tbody>
</table>
Class BoundReference

**ServletContextHelper** (org.osgi.framework.Bundle b)

Construct a new context helper and set the bundle associated with this context.

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>getMimeType</strong> (String name)</td>
<td>32</td>
</tr>
<tr>
<td>Maps a name to a MIME type.</td>
<td></td>
</tr>
<tr>
<td><strong>getRealPath</strong> (String path)</td>
<td>32</td>
</tr>
<tr>
<td>Gets the real path corresponding to the given virtual path.</td>
<td></td>
</tr>
<tr>
<td><strong>getResource</strong> (String name)</td>
<td>31</td>
</tr>
<tr>
<td>Maps a resource name to a URL.</td>
<td></td>
</tr>
<tr>
<td><strong>getResourcePaths</strong> (String path)</td>
<td>32</td>
</tr>
<tr>
<td>Returns a directory-like listing of all the paths to resources within the web application whose longest sub-path matches the supplied path argument.</td>
<td></td>
</tr>
<tr>
<td><strong>handleSecurity</strong> (HttpServletRequest request, HttpServletResponse response)</td>
<td>31</td>
</tr>
<tr>
<td>Handles security for the specified request.</td>
<td></td>
</tr>
</tbody>
</table>

### Field Detail

#### REMOTE_USER

Public static final String **REMOTE_USER** = "org.osgi.service.http.authentication.remote.user"

HttpServletRequest attribute specifying the name of the authenticated user. The value of the attribute can be retrieved by HttpServletRequest.getRemoteUser. This attribute name is org.osgi.service.http.authentication.remote.user.

#### AUTHENTICATION_TYPE

Public static final String **AUTHENTICATION_TYPE** = "org.osgi.service.http.authentication.type"

HttpServletRequest attribute specifying the scheme used in authentication. The value of the attribute can be retrieved by HttpServletRequest.getAuthType. This attribute name is org.osgi.service.http.authentication.type.

#### AUTHORIZATION

Public static final String **AUTHORIZATION** = "org.osgi.service.useradmin.authorization"

HttpServletRequest attribute specifying the Authorization object obtained from the org.osgi.service.useradmin.UserAdmin service. The value of the attribute can be retrieved by HttpServletRequest.getAttribute(HttpContext.AUTHORIZATION). This attribute name is org.osgi.service.useradmin.authorization.

### Constructor Detail

**ServletContextHelper**

Public **ServletContextHelper()**

Default constructor
Class BoundReference

ServletContextHelper

```java
class ServletContextHelper {
public ServletContextHelper(org.osgi.framework.Bundle b) {
    Construct a new context helper and set the bundle associated with this context.

    Parameters:
    b - The bundle
}

public boolean handleSecurity(HttpServletRequest request, HttpServletResponse response) throws IOException {
    Handles security for the specified request.

    The Http Whiteboard Service calls this method prior to servicing the specified request. This method
    controls whether the request is processed in the normal manner or an error is returned.

    If the request requires authentication and the Authorization header in the request is missing or not
    acceptable, then this method should set the WWW-Authenticate header in the response object, set the
    status in the response object to Unauthorized(401) and return false. See also RFC 2617: HTTP

    If the request requires a secure connection and the `getScheme` method in the request does not return
    'https' or some other acceptable secure protocol, then this method should set the status in the response
    object to Forbidden(403) and return false.

    When this method returns false, the Http Whiteboard Service will send the response back to the client,
    thereby completing the request. When this method returns true, the Http Whiteboard Service will proceed
    with servicing the request.

    If the specified request has been authenticated, this method must set the AUTHENTICATION_TYPE
    request attribute to the type of authentication used, and the REMOTE_USER request attribute to the remote user
    (request attributes are set using the `setAttribute` method on the request). If this method does not
    perform any authentication, it must not set these attributes.

    If the authenticated user is also authorized to access certain resources, this method must set the
    AUTHORIZATION request attribute to the Authorization object obtained from the
    org.osgi.service.useradmin.UserAdmin service.

    The servlet responsible for servicing the specified request determines the authentication type and remote
    user by calling the `getAuthType` and `getRemoteUser` methods, respectively, on the request.

    Parameters:
    request - The HTTP request.
    response - The HTTP response.

    Returns:
    true if the request should be serviced, false if the request should not be serviced and Http
    Whiteboard Service will send the response back to the client.

    Throws:
    IOException - may be thrown by this method. If this occurs, the Http Whiteboard Service will
    terminate the request and close the socket.
}

public URL getResource(String name) {
    Maps a resource name to a URL.
}
```

OSGi Javadoc -- 23.09.12
Class BoundReference

Called by the Http Whiteboard Service to map the specified resource name to a URL. For servlets, Http Whiteboard Service will call this method to support the ServletContext methods getResource and getResourceAsStream. For resource servlets, Http Whiteboard Service will call this method to locate the named resource.

The context can control from where resources come. For example, the resource can be mapped to a file in the bundle's persistent storage area via BundleContext.getDataFile(name).toURL() or to a resource in the context's bundle via getClass().getResource(name)

Parameters:
name - The name of the requested resource.

Returns:
A URL that Http Whiteboard Service can use to read the resource or null if the resource does not exist.

getMimeType

public String getMimeType(String name)

Maps a name to a MIME type.

Called by the Http Whiteboard Service to determine the MIME type for the specified name. For whiteboard services, the Http Whiteboard Service will call this method to support the ServletContext method getMimeType. For resource servlets, the Http Whiteboard Service will call this method to determine the MIME type for the Content-Type header in the response.

Parameters:
name - The name for which to determine the MIME type.

Returns:
The MIME type (e.g. text/html) of the specified name or null to indicate that the Http Service should determine the MIME type itself.

getResourcePaths

public Set<String> getResourcePaths(String path)

Returns a directory-like listing of all the paths to resources within the web application whose longest sub-path matches the supplied path argument.

Called by the Http Whiteboard Service to support the ServletContext method getResourcePaths for whiteboard services.

Parameters:
path - the partial path used to match the resources, which must start with a /

Returns:
a Set containing the directory listing, or null if there are no resources in the web application whose path begins with the supplied path.

getRealPath

public String getRealPath(String path)

Gets the real path corresponding to the given virtual path.

Called by the Http Whiteboard Service to support the ServletContext method getRealPath for whiteboard services.

Parameters:
path - the virtual path to be translated to a real path
Returns:
the real path, or null if the translation cannot be performed
Package org.osgi.service.http.runtime

@org.osgi.annotation.versioning.Version(value="1.0")

Http Service Runtime Package Version 1.0.

See: Description

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpServiceRuntime</td>
<td>35</td>
</tr>
</tbody>
</table>

The HttpServiceRuntime service represents the runtime information of a Http (Whiteboard) Service implementation.

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpServiceRuntimeConstants</td>
<td>36</td>
</tr>
</tbody>
</table>

Defines standard names for Http Runtime Service constants.

Package org.osgi.service.http.runtime Description

Http Service Runtime 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.http.runtime; version="[1.0,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.service.http.runtime; version="[1.0,1.1)"
The `HttpServiceRuntime` service represents the runtime information of a Http (Whiteboard) Service implementation.

It provides access to DTOs representing the current state of the service.

The `HttpServiceRuntime` service must at least be registered with the `HttpServiceRuntimeConstants.HTTP_SERVICE_ENDPOINT_ATTRIBUTE` attribute.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RequestInfoDTO calculateRequestInfoDTO(String path)</code></td>
<td>35</td>
</tr>
<tr>
<td>- Return a request info DTO containing the services involved with processing a request for the given path.</td>
<td></td>
</tr>
<tr>
<td><code>RuntimeDTO getRuntimeDTO()</code></td>
<td>35</td>
</tr>
<tr>
<td>- Return the runtime DTO representing the current state.</td>
<td></td>
</tr>
</tbody>
</table>

### Method Detail

#### getRuntimeDTO

```java
RuntimeDTO getRuntimeDTO()
```

Return the runtime DTO representing the current state.

**Returns:**
- The runtime DTO

#### calculateRequestInfoDTO

```java
RequestInfoDTO calculateRequestInfoDTO(String path)
```

Return a request info DTO containing the services involved with processing a request for the given path.

**Parameters:**
- path - The request path, relative to the root of the Http (Whiteboard) Service.

**Returns:**
- A request info DTO
Class HttpResponseMessage

generic
extends Object

final public class HttpResponseMessage
extends Object

Defines standard names for Http Runtime Service constants.

Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| HTTP_SERVICE_ENDPOINT_ATTRIBUTE | String | Http service runtime registration property specifying the endpoints upon which the Http service runtime is listening.

Field Detail

HTTP_SERVICE_ENDPOINT_ATTRIBUTE

class static final String HTTP_SERVICE_ENDPOINT_ATTRIBUTE = "osgi.http.endpoint"

Http service runtime registration property specifying the endpoints upon which the Http service runtime is listening.

An endpoint value is a URL or a relative path, to which the Http service runtime is listening. For example, http://192.168.1.10:8080/ or /myapp/. A relative path may be used if the scheme and authority parts of the URL are not known, e.g. in a bridged Http Service implementation. If the Http Service implementation is serving the root context and neither scheme nor authority is known, the value of the property is "/". Both, a URL and a relative path, must end with a slash.

An Http Service Runtime can be listening on multiple endpoints.

The value of this attribute must be of type String, String[], or Collection<String>.
Package org.osgi.service.http.runtime.dto

@org.osgi.annotation.versioning.Version(value="1.0")

Http Service Runtime DTO Package Version 1.0.

See:  

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseServletDTO</td>
<td>38</td>
</tr>
<tr>
<td>DTOConstants</td>
<td>40</td>
</tr>
<tr>
<td>ErrorPageDTO</td>
<td>42</td>
</tr>
<tr>
<td>FailedErrorPageDTO</td>
<td>44</td>
</tr>
<tr>
<td>FailedFilterDTO</td>
<td>46</td>
</tr>
<tr>
<td>FailedListenerDTO</td>
<td>47</td>
</tr>
<tr>
<td>FailedResourceDTO</td>
<td>48</td>
</tr>
<tr>
<td>FailedServletContextDTO</td>
<td>49</td>
</tr>
<tr>
<td>FailedServletDTO</td>
<td>51</td>
</tr>
<tr>
<td>FilterDTO</td>
<td>53</td>
</tr>
<tr>
<td>ListenerDTO</td>
<td>56</td>
</tr>
<tr>
<td>RequestInfoDTO</td>
<td>58</td>
</tr>
<tr>
<td>ResourceDTO</td>
<td>60</td>
</tr>
<tr>
<td>RuntimeDTO</td>
<td>62</td>
</tr>
<tr>
<td>ServletContextDTO</td>
<td>65</td>
</tr>
<tr>
<td>ServletDTO</td>
<td>68</td>
</tr>
</tbody>
</table>

Represents common information about a javax.servlet.Servlet service.  
Represents standard constants for the DTOS.  
Represents a javax.servlet.Servlet servlet for handling errors and currently being used by a servlet context.  
Represents a javax.servlet.Servlet service registered as an error page but currently not being used for a servlet context due to a problem.  
Represents a servlet Filter filter which is currently not being used for a servlet context due to a problem.  
Represents a listener service which is currently not being used for a servlet context due to a problem.  
Represents a resource definition which is currently not being used for a servlet context due to a problem.  
Represents a javax.servlet.ServletContext servlet context that is currently not used due to some problem.  
Represents a javax.servlet.Servlet servlet which is currently not being used for a servlet context due to a problem.  
Represents a servlet javax.servlet.Filter filter currently being used for a servlet context.  
Represents a listener currently being used for a servlet context.  
Represents the services used to process a specific request.  
Represents a resource definition currently being used for a servlet context.  
Represents the state of a Http Service Runtime.  
Represents a javax.servlet.ServletContext created for used servlets, resources, servlet filters, and listeners.  
Represents a javax.servlet.Servlet currently being used for a servlet context.  

Package org.osgi.service.http.runtime.dto Description

Http Service Runtime DTO 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.http.runtime.dto; version="[1.0,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.service.http.runtime.dto; version="[1.0,1.1)"
**Class BaseServletDTO**

```
org.osgi.service.http.runtime.dto
```

`java.lang.Object`

```
  org.osgi.dto.DTO
  org.osgi.service.http.runtime.dto.BaseServletDTO
```

**Direct Known Subclasses:**

- ErrorPageDTO
- ServletDTO

abstract public class BaseServletDTO
extends org.osgi.dto.DTO

Represents common information about a `javax.servlet.Servlet` service.

NotThreadSafe

### Field Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>boolean</code></td>
<td><code>asyncSupported</code></td>
<td>Specifies whether the servlet supports asynchronous processing.</td>
<td>39</td>
</tr>
<tr>
<td><code>Map&lt;String,String&gt;</code></td>
<td><code>initParams</code></td>
<td>The servlet initialization parameters as provided during registration of the servlet.</td>
<td>39</td>
</tr>
<tr>
<td><code>String</code></td>
<td><code>name</code></td>
<td>The name of the servlet.</td>
<td>38</td>
</tr>
<tr>
<td><code>long</code></td>
<td><code>serviceId</code></td>
<td>Service property identifying the servlet.</td>
<td>39</td>
</tr>
<tr>
<td><code>long</code></td>
<td><code>servletContextId</code></td>
<td>The service id of the <code>ServletContext</code> for the servlet.</td>
<td>39</td>
</tr>
<tr>
<td><code>String</code></td>
<td><code>servletInfo</code></td>
<td>The information string from the servlet.</td>
<td>38</td>
</tr>
</tbody>
</table>

### Constructor Summary

```
BaseServletDTO()  
```

39

Methods inherited from class org.osgi.dto.DTO

`toString`

### Field Detail

**name**

```
public String name
```

The name of the servlet.

**servletInfo**

```
public String servletInfo
```

The information string from the servlet.
This is the value returned by the Servlet.getServletInfo() method.

**asyncSupported**

```java
public boolean asyncSupported
```

Specifies whether the servlet supports asynchronous processing.

**initParams**

```java
public Map<String,String> initParams
```

The servlet initialization parameters as provided during registration of the servlet. Additional parameters like the Http Service Runtime attributes are not included.

**servletContextId**

```java
public long servletContextId
```

The service id of the ServletContext for the servlet.

**serviceId**

```java
public long serviceId
```

Service property identifying the servlet. In the case of a servlet registered in the service registry and picked up by a whiteboard implementation, this value is not negative and corresponds to the service id in the registry. If the servlet has not been registered in the service registry, the value is negative and a unique negative value is generated by the Http Service Runtime in this case.

**Constructor Detail**

**BaseServletDTO**

```java
public BaseServletDTO()
```
final public class DTOConstants extends Object

Defines standard constants for the DTOs.

Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAILURE_REASON_EXCEPTION_ON_INIT</td>
<td>0</td>
<td>An exception occurred during initializing of the service.</td>
</tr>
<tr>
<td>FAILURE_REASON_NO_SERVLET_CONTEXT_MATCHING</td>
<td>1</td>
<td>No matching servlet context.</td>
</tr>
<tr>
<td>FAILURE_REASON_SERVICE_ALREADY_USED</td>
<td>0</td>
<td>The service is not registered as a prototype scoped service and is already used with one servlet context and therefore can't be used with another servlet context.</td>
</tr>
<tr>
<td>FAILURE_REASON_SERVICE_NOT_GETTABLE</td>
<td>0</td>
<td>The service is registered in the servlet registry but getting the service fails as it returns null.</td>
</tr>
<tr>
<td>FAILURE_REASON_SERVLET_CONTEXT_FAILURE</td>
<td>0</td>
<td>Matching servlet context, but servlet context is not used due to a problem with the context.</td>
</tr>
<tr>
<td>FAILURE_REASON_SHADOWED_BY_OTHER_SERVICE</td>
<td>0</td>
<td>Service is shadowed by another service, e.g. a service with the same registration properties but a higher service ranking.</td>
</tr>
<tr>
<td>FAILURE_REASON_UNKNOWN</td>
<td>0</td>
<td>Failure reason is unknown</td>
</tr>
<tr>
<td>FAILURE_REASON_VALIDATION_FAILED</td>
<td>0</td>
<td>The service is registered in the servlet registry but the provided registration properties are invalid.</td>
</tr>
</tbody>
</table>

Field Detail

**FAILURE_REASON_UNKNOWN**

public static final int FAILURE_REASON_UNKNOWN = 0

Failure reason is unknown

The value of FAILURE_REASON_UNKNOWN is 0.

**FAILURE_REASON_NO_SERVLET_CONTEXT_MATCHING**

public static final int FAILURE_REASON_NO_SERVLET_CONTEXT_MATCHING = 1

No matching servlet context
The value of `FAILURE_REASON_NO_SERVLET_CONTEXT_MATCHING` is 1.

---

**FAILURE_REASON_SERVLET_CONTEXT_FAILURE**

```java
public static final int FAILURE_REASON_SERVLET_CONTEXT_FAILURE = 2
```

Matching servlet context, but servlet context is not used due to a problem with the context.

The value of `FAILURE_REASON_SERVLET_CONTEXT_FAILURE` is 2.

---

**FAILURE_REASON_SHADOWED_BY_OTHER_SERVICE**

```java
public static final int FAILURE_REASON_SHADOWED_BY_OTHER_SERVICE = 3
```

Service is shadowed by another service, e.g. a service with the same registration properties but a higher service ranking.

The value of `FAILURE_REASON_SHADOWED_BY_OTHER_SERVICE` is 3.

---

**FAILURE_REASON_EXCEPTION_ON_INIT**

```java
public static final int FAILURE_REASON_EXCEPTION_ON_INIT = 4
```

An exception occurred during initializing of the service. This reason can only happen for servlets and servlet filters.

The value of `FAILURE_REASON_EXCEPTION_ON_INIT` is 4.

---

**FAILURE_REASON_SERVICE_NOT_GETTABLE**

```java
public static final int FAILURE_REASON_SERVICE_NOT_GETTABLE = 5
```

The service is registered in the servlet registry but getting the service fails as it returns `null`.

The value of `FAILURE_REASON_SERVICE_NOT_GETTABLE` is 5.

---

**FAILURE_REASON_VALIDATION_FAILED**

```java
public static final int FAILURE_REASON_VALIDATION_FAILED = 6
```

The service is registered in the servlet registry but the provided registration properties are invalid.

The value of `FAILURE_REASON_VALIDATION_FAILED` is 6.

---

**FAILURE_REASON_SERVICE_ALREADY_USED**

```java
public static final int FAILURE_REASON_SERVICE_ALREADY_USED = 7
```

The service is not registered as a prototype scoped service and is already used with one servlet context and therefore can't be used with another servlet context.

The value of `FAILURE_REASON_SERVICE_ALREADY_USED` is 7.
Class ErrorPageDTO

org.osgi.service.http.runtime.dto

java.lang.Object
  org.osgi.dto.DTO
   org.osgi.service.http.runtime.dto.BaseServletDTO
    org.osgi.service.http.runtime.dto.ErrorPageDTO

Direct Known Subclasses:
  FailedErrorPageDTO

Public class ErrorPageDTO extends BaseServletDTO

Represents a javax.servlet.Servlet servlet for handling errors and currently being used by a servlet context.

NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>long[] errorCodes</td>
<td></td>
<td>The error codes the error page is used for.</td>
<td>42</td>
</tr>
<tr>
<td>String[] exceptions</td>
<td></td>
<td>The exceptions the error page is used for.</td>
<td>42</td>
</tr>
</tbody>
</table>

Fields inherited from class org.osgi.service.http.runtime.dto.BaseServletDTO
asyncSupported, initParams, name, serviceId, servletContextId, servletInfo

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorPageDTO()</td>
<td></td>
<td>43</td>
</tr>
</tbody>
</table>

Methods inherited from class org.osgi.dto.DTO

toString

toString

Field Detail

exceptions

public String[] exceptions

The exceptions the error page is used for. This array might be empty.

erroCodes

public long[] erroCodes

The error codes the error page is used for. This array might be empty.
Class ServletContextDTO

Constructor Detail

ErrorPageDTO

public ErrorPageDTO()
public class FailedErrorPageDTO extends ErrorPageDTO

Represents a javax.servlet.Servlet service registered as an error page but currently not being used for a servlet context due to a problem.

As the servlet represented by this DTO is not used due to a failure, the field `BaseServletDTO.servletContextId` always returns 0 and does not point to an existing servlet context.

NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failureReason</td>
<td>int</td>
<td>The reason why the servlet represented by this DTO is not used.</td>
</tr>
</tbody>
</table>

Fields inherited from class org.osgi.service.http.runtime.dto.ErrorPageDTO: errorCodes, exceptions

Fields inherited from class org.osgi.service.http.runtime.dto.BaseServletDTO: asyncSupported, initParams, name, serviceId, servletContextId, servletInfo

Constructor Summary

FailedErrorPageDTO()  

Methods inherited from class org.osgi.dto.DTO: toString

Field Detail

failureReason

public int failureReason

The reason why the servlet represented by this DTO is not used.

See Also: DTOConstants.FAILURE_REASON_UNKNOWN, DTOConstants.FAILURE_REASON_EXCEPTION_ON_INIT, DTOConstants.FAILURE_REASON_NO_SERVLET_CONTEXT_MATCHING, DTOConstants.FAILURE_REASON_SERVICE_NOT_GETTABLE, DTOConstants.FAILURE_REASON_SERVLET_CONTEXT_FAILURE, DTOConstants.FAILURE_REASON_SHADOWED_BY_OTHER_SERVICE
FailedErrorPageDTO

public FailedErrorPageDTO()
public class FailedFilterDTO
extends FilterDTO

Represents a servlet Filter filter which is currently not being used for a servlet context due to a problem.

As the service represented by this DTO is not used due to a failure, the field `FilterDTO.servletContextId` always returns 0 and does not point to an existing servlet context.

NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>int failureReason</th>
<th>The reason why the filter represented by this DTO is not used.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Page 46</td>
</tr>
</tbody>
</table>

Fields inherited from class org.osgi.service.http.runtime.dto.FilterDTO
asyncSupported, dispatcher, initParams, name, patterns, regexs, serviceId, servletContextId, servletNames

Constructor Summary

FailedFilterDTO()

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

failureReason

public int failureReason

The reason why the filter represented by this DTO is not used.

See Also:

DTOConstants.FAILURE_REASON_UNKNOWN, DTOConstants.FAILURE_REASON_EXCEPTION_ON_INIT,
DTOConstants.FAILURE_REASON_NO_SERVLET_CONTEXT_MATCHING,
DTOConstants.FAILURE_REASON_SERVICE_NOT_GETTABLE,
DTOConstants.FAILURE_REASON_SERVLET_CONTEXT_FAILURE,
DTOConstants.FAILURE_REASON_SHADOWED_BY_OTHER_SERVICE

Constructor Detail

FailedFilterDTO

public FailedFilterDTO()
Class FailedListenerDTO

org.osgi.service.http.runtime.dto

java.lang.Object
  org.osgi.dto.DTO
    org.osgi.service.http.runtime.dto.ListenerDTO
      org.osgi.service.http.runtime.dto.FailedListenerDTO

public class FailedListenerDTO
extends ListenerDTO

Represents a listener service which is currently not being used for a servlet context due to a problem.

As the listener represented by this DTO is not used due to a failure, the field `BaseServletDTO.servletContextId` always returns 0 and does not point to an existing servlet context.

NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>failureReason</td>
<td>int</td>
</tr>
</tbody>
</table>

The reason why the listener represented by this DTO is not used.

See Also: `DTOConstants.FAILURE_REASON_UNKNOWN`, `DTOConstants.FAILURE_REASON_EXCEPTION_ON_INIT`, `DTOConstants.FAILURE_REASON_NO_SERVLET_CONTEXT_MATCHING`, `DTOConstants.FAILURE_REASON_SERVICE NOT GETTABLE`, `DTOConstants.FAILURE_REASON_SERVLET_CONTEXT_FAILURE`, `DTOConstants.FAILURE_REASON_SHADOWED BY OTHER SERVICE`

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FailedListenerDTO()</td>
<td>47</td>
</tr>
</tbody>
</table>

Methods inherited from class org.osgi.dto.DTO

toString

Field Detail

failureReason

public int failureReason

The reason why the listener represented by this DTO is not used.

See Also: `DTOConstants.FAILURE_REASON_UNKNOWN`, `DTOConstants.FAILURE_REASON_EXCEPTION_ON_INIT`, `DTOConstants.FAILURE_REASON_NO_SERVLET_CONTEXT_MATCHING`, `DTOConstants.FAILURE_REASON_SERVICE NOT GETTABLE`, `DTOConstants.FAILURE_REASON_SERVLET_CONTEXT_FAILURE`, `DTOConstants.FAILURE_REASON_SHADOWED BY OTHER SERVICE`

Constructor Detail

FailedListenerDTO

public FailedListenerDTO()
public class FailedResourceDTO
extends ResourceDTO

Represents a resource definition which is currently not being used for a servlet context due to a problem.

As the service represented by this DTO is not used due to a failure, the field ResourceDTO.servletContextId always returns 0 and does not point to an existing servlet context.

NotThreadSafe

Field Detail

failureReason

public int failureReason

The reason why the resource represented by this DTO is not used.

See Also:
DTOConstants.FAILURE_REASON_UNKNOWN, DTOConstants.FAILURE_REASON_EXCEPTION_ON_INIT,
DTOConstants.FAILURE_REASON_NO_SERVLET_CONTEXT_MATCHING,
DTOConstants.FAILURE_REASON_SERVICE_NOT_GETTABLE,
DTOConstants.FAILURE_REASON_SERVLET_CONTEXT_FAILURE,
DTOConstants.FAILURE_REASON_SHADOWED_BY_OTHER_SERVICE

Constructor Detail

FailedResourceDTO

public FailedResourceDTO()
Class FailedServletContextDTO

Represents a javax.servlet.ServletContext servlet context that is currently not used due to some problem. The following fields return an empty array for a FailedServletContextDTO:

- ServletContextDTO.servletDTOs
- ServletContextDTO.resourceDTOs
- ServletContextDTO.filterDTOs
- ServletContextDTO.errorPageDTOs
- ServletContextDTO.listenerDTOs

The method ServletContextDTO.attributes returns an empty map for a FailedServletContextDTO.

NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>failureReason</td>
<td>int</td>
<td>The reason why the servlet context represented by this DTO is not used.</td>
<td>49</td>
</tr>
</tbody>
</table>

Fields inherited from class org.osgi.service.http.runtime.dto.ServletContextDTO

attributes, contextName, contextPath, errorPageDTOs, filterDTOs, initParams, listenerDTOs, names, resourceDTOs, serviceId, servletDTOs

Constructor Summary

FailedServletContextDTO()

Methods inherited from class org.osgi.dto.DTO

toString

Field Detail

failureReason

public int failureReason

The reason why the servlet context represented by this DTO is not used.

See Also:

DTOConstants.FAILURE_REASON_UNKNOWN, DTOConstants.FAILURE_REASON_EXCEPTION_ON_INIT, DTOConstants.FAILURE_REASON_NO_SERVLET_CONTEXT_MATCHING, DTOConstants.FAILURE_REASON_SERVICE_NOT_GETTABLE, DTOConstants.FAILURE_REASON_SERVLET_CONTEXT_FAILURE, DTOConstants.FAILURE_REASON_SHADOWED_BY_OTHER_SERVICE
Constructor Detail

FailedServletContextDTO

public FailedServletContextDTO()
Class FailedServletDTO

java.lang.Object
  org.osgi.dto.DTO
    org.osgi.service.http.runtime.dto.BaseServletDTO
      org.osgi.service.http.runtime.dto.ServletDTO
       org.osgi.service.http.runtime.dto.FailedServletDTO

public class FailedServletDTO
  extends ServletDTO

Represents a javax.servlet.Servlet servlet which is currently not being used for a servlet context due to a problem.

As the servlet represented by this DTO is not used due to a failure, the field BaseServiceDTO.servletContextId always returns 0 and does not point to an existing servlet context.

NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>failureReason</td>
<td>51</td>
</tr>
</tbody>
</table>

The reason why the servlet represented by this DTO is not used.

Fields inherited from class org.osgi.service.http.runtime.dto.ServletDTO

- patterns

Fields inherited from class org.osgi.service.http.runtime.dto.BaseServletDTO

- asyncSupported, initParams, name, serviceId, servletContextId, servletInfo

Constructor Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FailedServletDTO()</td>
<td>52</td>
</tr>
</tbody>
</table>

Methods inherited from class org.osgi.dto.DTO

- toString

Field Detail

failureReason

public int failureReason

The reason why the servlet represented by this DTO is not used.

See Also:

- DTOConstants.FAILURE_REASON_UNKNOWN, DTOConstants.FAILURE_REASON_EXCEPTION_ON_INIT,
- DTOConstants.FAILURE_REASON_NO_SERVLET_CONTEXT_MATCHING,
- DTOConstants.FAILURE_REASON_SERVICE_NOT_GETTABLE,
- DTOConstants.FAILURE_REASON_SERVLET_CONTEXT_FAILURE,
- DTOConstants.FAILURE_REASON_SHADOWED_BY_OTHER_SERVICE
Constructor Detail

FailedServletDTO

public FailedServletDTO()
Class FilterDTO

org.osgi.service.http.runtime.dto

java.lang.Object

- org.osgi.dto.DTO
  - org.osgi.service.http.runtime.dto.FilterDTO

Direct Known Subclasses:
- FailedFilterDTO

public class FilterDTO
extends org.osgi.dto.DTO

Represents a servlet javax.servlet.Filter filter currently being used for a servlet context.

NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asyncSupported</td>
<td>boolean</td>
<td>Specifies whether the servlet filter supports asynchronous processing.</td>
</tr>
<tr>
<td>dispatcher</td>
<td>String[]</td>
<td>The dispatcher associations for the servlet filter.</td>
</tr>
<tr>
<td>initParams</td>
<td>Map&lt;String, String&gt;</td>
<td>The filter initialization parameters as provided during registration of the filter.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>The name of the servlet filter.</td>
</tr>
<tr>
<td>patterns</td>
<td>String[]</td>
<td>The request mappings for the servlet filter.</td>
</tr>
<tr>
<td>regexs</td>
<td>String[]</td>
<td>The request mappings for the servlet filter.</td>
</tr>
<tr>
<td>serviceId</td>
<td>long</td>
<td>Service property identifying the filter.</td>
</tr>
<tr>
<td>servletContextId</td>
<td>long</td>
<td>The service id of the ServletContext for the servlet filter.</td>
</tr>
<tr>
<td>servletNames</td>
<td>String[]</td>
<td>The servlet names for the servlet filter.</td>
</tr>
</tbody>
</table>

Constructor Summary

FilterDTO ()

Methods inherited from class org.osgi.dto.DTO

toString

Field Detail

name

public String name

The name of the servlet filter.
patterns

public String[] patterns

The request mappings for the servlet filter.
The specified patterns are used to determine whether a request is mapped to the servlet filter.

servletNames

public String[] servletNames

The servlet names for the servlet filter.
The specified names are used to determine the servlets whose requests are mapped to the servlet filter.

regexs

public String[] regexs

The request mappings for the servlet filter.
The specified regular expressions are used to determine whether a request is mapped to the servlet filter.

asyncSupported

public boolean asyncSupported

Specifies whether the servlet filter supports asynchronous processing.

dispatcher

public String[] dispatcher

The dispatcher associations for the servlet filter.
The specified names are used to determine in what occasions the servlet filter is called.

initParams

public Map<String, String> initParams

The filter initialization parameters as provided during registration of the filter. Additional parameters like the Http Service Runtime attributes are not included.

serviceId

public long serviceId

Service property identifying the filter. In the case of a filter registered in the service registry and picked up by a whiteboard implementation, this value is not negative and corresponds to the service id in the registry.
If the filter has not been registered in the service registry, the value is negative and a unique negative value is generated by the Http Service Runtime in this case.

**servletContextId**

```java
public long servletContextId
```

The service id of the `ServletContext` for the servlet filter.

### Constructor Detail

**FilterDTO**

```java
public FilterDTO()
```
Class ListenerDTO

org.osgi.service.http.runtime.dto

java.lang.Object
  org.osgi.dto.DTO
   org.osgi.service.http.runtime.dto.ListenerDTO

Direct Known Subclasses:
  FailedListenerDTO

public class ListenerDTO
extends org.osgi.dto.DTO

Represents a listener currently being used for a servlet context.

NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>serviceId</td>
<td>Service property identifying the listener.</td>
<td>56</td>
</tr>
<tr>
<td>long</td>
<td>servletContextId</td>
<td>The service id of the ServletContext for the listener.</td>
<td>57</td>
</tr>
<tr>
<td>String[]</td>
<td>types</td>
<td>The fully qualified type names the listener.</td>
<td>56</td>
</tr>
</tbody>
</table>

Constructor Summary

ListenerDTO() | 57

Methods inherited from class org.osgi.dto.DTO

toString

Field Detail

types

public String[] types

The fully qualified type names the listener.

serviceId

public long serviceId

Service property identifying the listener. In the case of a listener registered in the service registry and picked up by a whiteboard implementation, this value is not negative and corresponds to the service id in the registry. If the listener has not been registered in the service registry, the value is negative and a unique negative value is generated by the Http Service Runtime in this case.
Class FilterDTO

servletContextId

public long servletContextId

    The service id of the ServletContext for the listener.

Constructor Detail

ListenerDTO

public ListenerDTO()
Class RequestInfoDTO

org.osgi.service.http.runtime.dto

java.lang.Object
   | org.osgi.dto.DTO
   | org.osgi.service.http.runtime.dto.RequestInfoDTO

public class RequestInfoDTO
extends org.osgi.dto.DTO

Represents the services used to process a specific request.

NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FilterDTO[]</td>
<td>filterDTOs</td>
<td>The filters processing this request.</td>
<td>58</td>
</tr>
<tr>
<td>String</td>
<td>path</td>
<td>The path of the request relative to the root.</td>
<td>58</td>
</tr>
<tr>
<td>ResourceDTO</td>
<td>resourceDTO</td>
<td>The resource processing this request.</td>
<td>59</td>
</tr>
<tr>
<td>long</td>
<td>servletContextId</td>
<td>The service id of the ServletContext for this request.</td>
<td>58</td>
</tr>
<tr>
<td>ServletDTO</td>
<td>servletDTO</td>
<td>The servlet processing this request.</td>
<td>59</td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestInfoDTO()</td>
<td>59</td>
</tr>
</tbody>
</table>

Methods inherited from class org.osgi.dto.DTO

toString

Field Detail

path

public String path

   The path of the request relative to the root.

servletContextId

public long servletContextId

   The service id of the ServletContext for this request.

filterDTOS

public FilterDTO[] filterDTOs
Interface HttpServiceRuntime

The filters processing this request.

**servletDTO**

```java
public ServletDTO servletDTO
```

The servlet processing this request. If the request is processed by a servlet, this field points to the DTO of the servlet. If the request is processed by another type of component like a resource, this field is null.

**resourceDTO**

```java
public ResourceDTO resourceDTO
```

The resource processing this request. If the request is processed by a resource, this field points to the DTO of the resource. If the request is processed by another type of component like a servlet, this field is null.

### Constructor Detail

#### RequestInfoDTO

```java
public RequestInfoDTO()
```
Class ResourceDTO

org.osgi.service.http.runtime.dto

java.lang.Object
  org.osgi.dto.DTO
  org.osgi.service.http.runtime.dto.ResourceDTO

Direct Known Subclasses:
  FailedResourceDTO

public class ResourceDTO
extends org.osgi.dto.DTO

Represents a resource definition currently being used for a servlet context.

NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>patterns</td>
<td>String[]</td>
<td>The request mappings for the resource.</td>
<td>60</td>
</tr>
<tr>
<td>prefix</td>
<td>String</td>
<td>The prefix of the resource.</td>
<td>60</td>
</tr>
<tr>
<td>serviceId</td>
<td>long</td>
<td>Service property identifying the resource.</td>
<td>61</td>
</tr>
<tr>
<td>servletContextId</td>
<td>long</td>
<td>The service id of the ServletContext for the resource.</td>
<td>61</td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceDTO()</td>
<td></td>
<td>61</td>
</tr>
</tbody>
</table>

Methods inherited from class org.osgi.dto.DTO

toString

Field Detail

patterns

public String[] patterns

The request mappings for the resource.

The specified patterns are used to determine whether a request is mapped to the resource.

prefix

public String prefix

The prefix of the resource.
**serviceId**

public long serviceId

Service property identifying the resource. In the case of a resource registered in the service registry and picked up by a whiteboard implementation, this value is not negative and corresponds to the service id in the registry. If the resource has not been registered in the service registry, the value is negative and a unique negative value is generated by the Http Service Runtime in this case.

---

**servletContextId**

public long servletContextId

The service id of the ServletContext for the resource.

---

**Constructor Detail**

**ResourceDTO**

public ResourceDTO()
public class RuntimeDTO
extends org.osgi.dto.DTO

Represents the state of a Http Service Runtime.

Field Summary

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attributes</strong></td>
<td>62</td>
</tr>
<tr>
<td>The runtime attributes.</td>
<td></td>
</tr>
<tr>
<td><strong>failedErrorPageDTOs</strong></td>
<td>63</td>
</tr>
<tr>
<td>Returns the representations of the error page javax.servlet.ServletExceptions associated with this runtime but currently not used due to some problem.</td>
<td></td>
</tr>
<tr>
<td><strong>failedFilterDTOs</strong></td>
<td>63</td>
</tr>
<tr>
<td>Returns the representations of the servlet javax.servlet.Filter filters associated with this runtime but currently not used due to some problem.</td>
<td></td>
</tr>
<tr>
<td><strong>failedListenerDTOs</strong></td>
<td>63</td>
</tr>
<tr>
<td>Returns the representations of the listeners associated with this runtime but currently not used due to some problem.</td>
<td></td>
</tr>
<tr>
<td><strong>failedResourceDTOs</strong></td>
<td>63</td>
</tr>
<tr>
<td>Returns the representations of the resources associated with this runtime but currently not used due to some problem.</td>
<td></td>
</tr>
<tr>
<td><strong>failedServletContextDTOs</strong></td>
<td>63</td>
</tr>
<tr>
<td>Returns the representations of the javax.servlet.ServletContext objects currently not used by the Http service runtime due to some problem.</td>
<td></td>
</tr>
<tr>
<td><strong>failedServletDTOs</strong></td>
<td>63</td>
</tr>
<tr>
<td>Returns the representations of the javax.servlet.Servlets associated with this runtime but currently not used due to some problem.</td>
<td></td>
</tr>
<tr>
<td><strong>servletContextDTOs</strong></td>
<td>63</td>
</tr>
<tr>
<td>Returns the representations of the javax.servlet.ServletContext objects used by the Http service runtime.</td>
<td></td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RuntimeDTO ()</td>
<td>64</td>
</tr>
</tbody>
</table>

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

attributes

public Map<String, String> attributes

The runtime attributes.
Class ListenerDTO

**servletContextDTOs**

```java
public ServletContextDTO[] servletContextDTOs
```

Returns the representations of the `javax.servlet.ServletContext` objects used by the Http service runtime. The returned array may be empty if the Http service runtime is currently not using any `ServletContext` objects.

**failedServletContextDTOs**

```java
public FailedServletContextDTO[] failedServletContextDTOs
```

Returns the representations of the `javax.servlet.ServletException` objects currently not used by the Http service runtime due to some problem. The returned array may be empty.

**failedServletDTOs**

```java
public FailedServletDTO[] failedServletDTOs
```

Returns the representations of the `javax.servlet.Servlet` servlets associated with this runtime but currently not used due to some problem. The returned array may be empty.

**failedResourceDTOs**

```java
public FailedResourceDTO[] failedResourceDTOs
```

Returns the representations of the resources associated with this runtime but currently not used due to some problem. The returned array may be empty.

**failedFilterDTOs**

```java
public FailedFilterDTO[] failedFilterDTOs
```

Returns the representations of the servlet `javax.servlet.Filter` filters associated with this runtime but currently not used due to some problem. The returned array may be empty.

**failedErrorPageDTOs**

```java
public FailedErrorPageDTO[] failedErrorPageDTOs
```

Returns the representations of the error page `javax.servlet.Servlet` servlets associated with this runtime but currently not used due to some problem. The returned array may be empty.

**failedListenerDTOs**

```java
public FailedListenerDTO[] failedListenerDTOs
```

Returns the representations of the listeners associated with this runtime but currently not used due to some problem. The returned array may be empty.
Class ListenerDTO

Constructor Detail

RuntimeDTO

public RuntimeDTO()
Class ServletContextDTO

`org.osgi.service.http.runtime.dto`

java.lang.Object
   org.osgi.dto.DTO
   org.osgi.service.http.runtime.dto.ServletContextDTO

Direct Known Subclasses:
   FailedServletContextDTO

public class ServletContextDTO
extends org.osgi.dto.DTO

Represents a `javax.servlet.ServletContext` created for used servlets, resources, servlet filters, and listeners. The servlet context is backed by a `ServletContextHelper` service.

NotThreadSafe

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map&lt;String, Object&gt; <code>attributes</code></td>
<td>The servlet context attributes.</td>
</tr>
<tr>
<td>String <code>contextName</code></td>
<td>The context name of the servlet context.</td>
</tr>
<tr>
<td>String <code>contextPath</code></td>
<td>The servlet context path.</td>
</tr>
<tr>
<td>ErrorPageDTO[] <code>errorPageDTOs</code></td>
<td>Returns the representations of the error page <code>Servlet</code> services associated with this context.</td>
</tr>
<tr>
<td>FilterDTO[] <code>filterDTOs</code></td>
<td>Returns the representations of the servlet <code>Filter</code> services associated with this context.</td>
</tr>
<tr>
<td>Map&lt;String, String&gt; <code>initParams</code></td>
<td>The servlet context initialization parameters.</td>
</tr>
<tr>
<td>ListenerDTO[] <code>listenerDTOs</code></td>
<td>Returns the representations of the listener services associated with this context.</td>
</tr>
<tr>
<td>String[] <code>names</code></td>
<td>The names of the HTTP context.</td>
</tr>
<tr>
<td>ResourceDTO[] <code>resourceDTOs</code></td>
<td>Returns the representations of the resource services associated with this context.</td>
</tr>
<tr>
<td>long <code>serviceId</code></td>
<td>Service property identifying the servlet context.</td>
</tr>
<tr>
<td>ServletDTO[] <code>servletDTOs</code></td>
<td>Returns the representations of the <code>Servlet</code> services associated with this context.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServletContextDTO()</td>
<td>67</td>
</tr>
</tbody>
</table>

Methods inherited from class org.osgi.dto.DTO
toString
Field Detail

names

public String[] names

The names of the HTTP context. An array of the names the corresponding ServletContextHelper is used for.

contextName

public String contextName

The context name of the servlet context.

This is the value returned by the ServletContext.getServletContextName() method.

contextPath

public String contextPath

The servlet context path. This is the value returned by the ServletContext.getContextPath() method.

initParams

public Map<String,String> initParams

The servlet context initialization parameters. This is the set of parameters provided when registering this context. Additional parameters like the Http Service Runtime attributes are not included.

attributes

public Map<String,Object> attributes

The servlet context attributes.

The value type must be a numerical type, Boolean, String, DTO or an array of any of the former. Therefore this method will only return the attributes of the servlet context conforming to this constraint.

serviceId

public long serviceId

Service property identifying the servlet context. In the case of a servlet context registered in the service registry and picked up by a whiteboard implementation, this value is not negative and corresponds to the service id in the registry. If the servlet context has not been registered in the service registry, the value is negative and a unique negative value is generated by the Http Service Runtime in this case.

servletDTOs

public ServletDTO[] servletDTOs
Class ResourceDTO

Returns the representations of the Servlet services associated with this context. The representations of the Servlet services associated with this context. The returned array may be empty if this context is currently not associated with any Servlet services.

resourceDTOs

public ResourceDTO[] resourceDTOs

Returns the representations of the resource services associated with this context. The representations of the resource services associated with this context. The returned array may be empty if this context is currently not associated with any resource services.

filterDTOs

public FilterDTO[] filterDTOs

Returns the representations of the servlet Filter services associated with this context. The representations of the servlet Filter services associated with this context. The returned array may be empty if this context is currently not associated with any servlet Filter services.

errrorPageDTOs

public ErrorPageDTO[] errorPageDTOs

Returns the representations of the error page Servlet services associated with this context. The representations of the error page Servlet services associated with this context. The returned array may be empty if this context is currently not associated with any error pages.

listenerDTOs

public ListenerDTO[] listenerDTOs

Returns the representations of the listener services associated with this context. The representations of the listener services associated with this context. The returned array may be empty if this context is currently not associated with any listener services.

Constructor Detail

ServletContextDTO

public ServletContextDTO()
Class ServletDTO

```
java.lang.Object
└── org.osgi.dto.DTO
    └── org.osgi.service.http.runtime.dto.BaseServletDTO
        └── org.osgi.service.http.runtime.dto.ServletDTO

Direct Known Subclasses:
  FailedServletDTO
```

```java
public class ServletDTO extends BaseServletDTO

Represents a javax.servlet.Servlet currently being used for a servlet context.

NotThreadSafe
```

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String[] patterns</td>
<td>68</td>
</tr>
</tbody>
</table>

The request mappings for the servlet.

Fields inherited from class org.osgi.service.http.runtime.dto.BaseServletDTO

asyncSupported, initParams, name, serviceId, servletContextId, servletInfo

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServletDTO()</td>
<td>68</td>
</tr>
</tbody>
</table>

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

patterns

```java
public String[] patterns

The request mappings for the servlet.

The specified patterns are used to determine whether a request is mapped to the servlet.
```

Constructor Detail

ServletDTO

```java
public ServletDTO()```
Package org.osgi.service.http.whiteboard

@org.osgi.annotation.versioning.Version(value="1.0")

Http Service Whiteboard Package Version 1.0.

See: Description

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpWhiteboardConstants</td>
<td>70</td>
</tr>
<tr>
<td>Defines standard names for Http Whiteboard Service constants.</td>
<td></td>
</tr>
</tbody>
</table>

Package org.osgi.service.http.whiteboard Description

Http Service Whiteboard 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.http.whiteboard; version="[1.0,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.service.http.whiteboard; version="[1.0,1.1)"
Package org.osgi.service.http.whiteboard

Class HttpWhiteboardConstants

org.osgi.service.http.whiteboard

java.lang.Object

\_org.osgi.service.http.whiteboard.HttpWhiteboardConstants

final public class HttpWhiteboardConstants
extends Object

Defines standard names for Http Whiteboard Service constants.

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static String DISPATCHER_ASYNC Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied in the async context.</td>
<td>75</td>
</tr>
<tr>
<td>static String DISPATCHER_ERROR Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied when an error page is called.</td>
<td>75</td>
</tr>
<tr>
<td>static String DISPATCHER_FORWARD Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied to forward calls to the dispatcher.</td>
<td>75</td>
</tr>
<tr>
<td>static String DISPATCHER_INCLUDE Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied to include calls to the dispatcher.</td>
<td>75</td>
</tr>
<tr>
<td>static String DISPATCHER_REQUEST Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied to client requests.</td>
<td>74</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_CONTEXT_NAME Service property specifying the name(s) of an ServletContextHelper service.</td>
<td>71</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_CONTEXT_PATH Service property specifying the path of an ServletContextHelper service.</td>
<td>71</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_CONTEXT_SELECT Service property referencing the ServletContextHelper service.</td>
<td>72</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_DEFAULT_CONTEXT_NAME The name of the default ServletContextHelper.</td>
<td>71</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_FILTER_ASYNC_SUPPORTED Service property specifying whether a Filter service supports asynchronous processing.</td>
<td>74</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_FILTER_DISPATCHER Service property specifying the dispatcher handling of a Filter.</td>
<td>74</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_FILTER_NAME Service property specifying the servlet filter name of a Filter service.</td>
<td>73</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_FILTER_PATTERN Service property specifying the request mappings for a Filter service.</td>
<td>73</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_FILTER_REGEX Service property specifying the request mappings for a Filter service.</td>
<td>74</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_FILTER_SERVLET Service property specifying the servlet_names for a Filter service.</td>
<td>73</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_RESOURCE_PREFIX Service property specifying the resource entry prefix for a Servlet servlet service.</td>
<td>75</td>
</tr>
<tr>
<td>static String HTTP_WHITEBOARD_SERVLET_ASYNC_SUPPORTED Service property specifying whether a Servlet service supports asynchronous processing.</td>
<td>73</td>
</tr>
</tbody>
</table>
### Field Detail

#### HTTP_WHITEBOARD_DEFAULT_CONTEXT_NAME

```java
public static final String HTTP_WHITEBOARD_DEFAULT_CONTEXT_NAME = "default"
```

The name of the default [ServletContextHelper](#). If a service is registered with this property, it is overriding the default context with a custom provided context.

**See Also:** [HTTP_WHITEBOARD_CONTEXT_NAME](#)

#### HTTP_WHITEBOARD_CONTEXT_NAME

```java
public static final String HTTP_WHITEBOARD_CONTEXT_NAME = "osgi.http.whiteboard.context.name"
```

Service property specifying the name(s) of an [ServletContextHelper](#) service.

For [ServletContextHelper](#) services, this service property must be specified. Context services without this service property must be ignored.

Servlet, listener, servlet filter, and resource servlet services might refer to a specific [ServletContextHelper](#) service referencing the name with the [HTTP_WHITEBOARD_CONTEXT_SELECT](#) property.

For [ServletContextHelper](#) services, the value of this service property must be of type `String`, `String[]`, or `Collection<String>`. Each value must follow the "symbolic-name" specification from Section 1.3.2 of the OSGi Core Specification.

**See Also:**
- [HTTP_WHITEBOARD_CONTEXT_PATH](#),
- [HTTP_WHITEBOARD_DEFAULT_CONTEXT_NAME](#),
- [HTTP_WHITEBOARD_CONTEXT_SELECT](#)

#### HTTP_WHITEBOARD_CONTEXT_PATH

```java
public static final String HTTP_WHITEBOARD_CONTEXT_PATH = "osgi.http.whiteboard.context.path"
```

Service property specifying the path of an [ServletContextHelper](#) service.

For [ServletContextHelper](#) services this service property is required. Context services without this service property must be ignored.

This property defines a context path under which all whiteboard services are registered. Having different contexts with different paths allows to separate the URL space.

For [ServletContextHelper](#) services, the value of this service property must be of type `String`. The path must start with a slash but not end with a slash. Contexts with an invalid path are ignored.
HTTP_WHITEBOARD_CONTEXT_SELECT

```java
public static final String HTTP_WHITEBOARD_CONTEXT_SELECT =
    "osgi.http.whiteboard.context.select"
```

Service property referencing the `ServletContextHelper` service.

For servlet, listener, servlet filter, or resource servlet services, this service property refers to the associated Servlet Context Helper service. The value of this property either directly referencing a context name or is a filter expression which is matched against the service registration properties of the Servlet Context Helper. If this service property is not specified, then the default context must be used. If there is no context service matching, the servlet, listener, servlet filter, or resource servlet service must be ignored.

For servlet, listener, servlet filter, or resource servlet services, the value of this service property must be of type `String`.

See Also:
- `HTTP_WHITEBOARD_CONTEXT_NAME`
- `HTTP_WHITEBOARD_CONTEXT_PATH`

HTTP_WHITEBOARD_SERVLET_NAME

```java
public static final String HTTP_WHITEBOARD_SERVLET_NAME =
    "osgi.http.whiteboard.servlet.name"
```

Service property specifying the servlet name of a Servlet service.

This name is used as the value for the `ServletConfig.getServletName()` method. If this service property is not specified, the fully qualified name of the service object's class is used as the servlet name. Filter services may refer to servlets by this name in their `HTTP_WHITEBOARD_FILTER_SERVLET` service property to apply the filter to the servlet.

Servlet names must be unique among all servlet services associated with a `ServletContextHelper`. If multiple servlet services associated with the same HttpContext have the same servlet name, then all but the highest ranked servlet service must be ignored.

The value of this service property must be of type `String`.

HTTP_WHITEBOARD_SERVLET_PATTERN

```java
public static final String HTTP_WHITEBOARD_SERVLET_PATTERN =
    "osgi.http.whiteboard.servlet.pattern"
```

Service property specifying the request mappings for a Servlet service.

The specified patterns are used to determine whether a request should be mapped to the servlet. Servlet services without this service property or `HTTP_WHITEBOARD_SERVLET_ERROR_PAGE` must be ignored.

The value of this service property must be of type `String`, `String[]`, or `Collection<String>`.  

See Also:
- "Java Servlet Specification Version 3.0, Section 12.2 Specification of Mappings"

HTTP_WHITEBOARD_SERVLET_ERROR_PAGE

```java
public static final String HTTP_WHITEBOARD_SERVLET_ERROR_PAGE =
    "osgi.http.whiteboard.servlet.errorPage"
```

See Also:
- `HTTP_WHITEBOARD_CONTEXT_NAME`
- `HTTP_WHITEBOARD_CONTEXT_SELECT`
Service property specifying whether a Servlet service acts as an error page.

The service property values may be the name of a fully qualified exception class or a three digit HTTP status code. Any value that is not a three digit number is considered to be the name of a fully qualified exception class.

The value of this service property must be of type String, String[], or Collection<String>.

**HTTP_WHITEBOARD_SERVLET_ASYNC_SUPPORTED**

```java
public static final String HTTP_WHITEBOARD_SERVLET_ASYNC_SUPPORTED =
    "osgi.http.whiteboard.servlet.asyncSupported"
```

Service property specifying whether a Servlet service supports asynchronous processing.

By default Servlet services do not support asynchronous processing.

The value of this service property must be of type Boolean.

See Also:
"Java Servlet Specification Version 3.0, Section 2.3.3.3 Asynchronous Processing"

**HTTP_WHITEBOARD_FILTER_NAME**

```java
public static final String HTTP_WHITEBOARD_FILTER_NAME =
      "osgi.http.whiteboard.filter.name"
```

Service property specifying the servlet filter name of a Filter service.

This name is used as the value for the FilterConfig.getFilterName() method. If this service property is not specified, the fully qualified name of the service object's class is used as the servlet filter name.

Servlet filter names must be unique among all servlet filter services associated with an ServletContextHelper. If multiple servlet filter services associated with the same HttpContext have the same servlet filter name, then all but the highest ranked servlet filter service must be ignored.

The value of this service property must be of type String.

**HTTP_WHITEBOARD_FILTER_PATTERN**

```java
public static final String HTTP_WHITEBOARD_FILTER_PATTERN =
     "osgi.http.whiteboard.filter.pattern"
```

Service property specifying the request mappings for a Filter service.

The specified patterns are used to determine whether a request should be mapped to the servlet filter. Filter services without this service property or the HTTP_WHITEBOARD_FILTER_SERVLET or the HTTP_WHITEBOARD_FILTER_REGEX service property must be ignored.

The value of this service property must be of type String, String[], or Collection<String>.

See Also:
"Java Servlet Specification Version 3.0, Section 12.2 Specification of Mappings"

**HTTP_WHITEBOARD_FILTER_SERVLET**

```java
public static final String HTTP_WHITEBOARD_FILTER_SERVLET =
     "osgi.http.whiteboard.filter.servlet"
```
Service property specifying the `servlet_names` for a `Filter` service.

The specified names are used to determine the servlets whose requests should be mapped to the servlet filter. Filter services without this service property or the `HTTP_WHITEBOARD_FILTER_PATTERN` or the `HTTP_WHITEBOARD_FILTER_REGEX` service property must be ignored.

The value of this service property must be of type `String`, `String[]`, or `Collection<String>`.

---

**HTTP_WHITEBOARD_FILTER_REGEX**

```java
public static final String HTTP_WHITEBOARD_FILTER_REGEX = "osgi.http.whiteboard.filter.regex"
```

Service property specifying the request mappings for a `Filter` service.

The specified regular expressions are used to determine whether a request should be mapped to the servlet filter. The regular expressions must follow the syntax defined in . Filter services without this service property or the `HTTP_WHITEBOARD_FILTER_SERVLET` or the `HTTP_WHITEBOARD_FILTER_PATTERN` service property must be ignored.

The value of this service property must be of type `String`, `String[]`, or `Collection<String>`.

See Also:

`java.util.regex.Pattern`

---

**HTTP_WHITEBOARD_FILTER_ASYNC_SUPPORTED**

```java
public static final String HTTP_WHITEBOARD_FILTER_ASYNC_SUPPORTED = "osgi.http.whiteboard.filter.asyncSupported"
```

Service property specifying whether a `Filter` service supports asynchronous processing.

By default Filters services do not support asynchronous processing.

The value of this service property must be of type `Boolean`.

See Also:

"Java Servlet Specification Version 3.0, Section 2.3.3.3 Asynchronous Processing"

---

**HTTP_WHITEBOARD_FILTER_DISPATCHER**

```java
public static final String HTTP_WHITEBOARD_FILTER_DISPATCHER = "osgi.http.whiteboard.filter.dispatcher"
```

Service property specifying the dispatcher handling of a `Filter`.

By default Filters services are associated with client requests only (see value `DISPATCHER_REQUEST`).

The value of this service property must be of type `String`, `String[]`, or `Collection<String>`. Allowed values are `DISPATCHER_ASYNC`, `DISPATCHER_ERROR`, `DISPATCHER_FORWARD`, `DISPATCHER_INCLUDE`, `DISPATCHER_REQUEST`.

See Also:

"Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

---

**DISPATCHER_REQUEST**

```java
public static final String DISPATCHER_REQUEST = "REQUEST"
```
Possible value for the `HTTP_WHITEBOARD_FILTER_DISPATCHER` property indicating the filter is applied to client requests.

See Also: "Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

---

**DISPATCHER_INCLUDE**

```java
public static final String DISPATCHER_INCLUDE = "INCLUDE"
```

Possible value for the `HTTP_WHITEBOARD_FILTER_DISPATCHER` property indicating the filter is applied to include calls to the dispatcher.

See Also: "Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

---

**DISPATCHER_FORWARD**

```java
public static final String DISPATCHER_FORWARD = "FORWARD"
```

Possible value for the `HTTP_WHITEBOARD_FILTER_DISPATCHER` property indicating the filter is applied to forward calls to the dispatcher.

See Also: "Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

---

**DISPATCHER_ASYNC**

```java
public static final String DISPATCHER_ASYNC = "ASYNC"
```

Possible value for the `HTTP_WHITEBOARD_FILTER_DISPATCHER` property indicating the filter is applied in the async context.

See Also: "Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

---

**DISPATCHER_ERROR**

```java
public static final String DISPATCHER_ERROR = "ERROR"
```

Possible value for the `HTTP_WHITEBOARD_FILTER_DISPATCHER` property indicating the filter is applied when an error page is called.

See Also: "Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

---

**HTTP_WHITEBOARD_RESOURCE_PREFIX**

```java
public static final String HTTP_WHITEBOARD_RESOURCE_PREFIX = "osgi.http.whiteboard.resource.prefix"
```

Service property specifying the resource entry prefix for a Servlet servlet service.

If a servlet service is registered with this property, it is marked as a resource serving servlet serving bundle resources.
This prefix is used to map a requested resource to the bundle's entries. TODO do we distinguish between "xyz" and "xyz"?

The value of this service property must be of type String.

HTTP_WHITEBOARD_TARGET

public static final String HTTP_WHITEBOARD_TARGET = "osgi.http.whiteboard.target"

Service property specifying the target filter to select the Http Whiteboard Service runtime to process the service.

An Http Whiteboard Service implementation can define any number of attributes which can be referenced by the target filter. The attributes should always include the osgi.http.endpoint attribute if the endpoint information is known.

If this service property is not specified, then all Http Whiteboard Services can process the service.

The value of this service property must be of type String and be a valid filter string.

8 Considered Alternatives

8.1 New methods to register Servlets and Filters

In addition to the proposed support for Whiteboard style registration of Servlets, Filters, Resources, HttpContexts, and error pages the Http Service API could have been extended to support programmatic support for such registration.

At the CPEG F2F in Austin it was decided that we should only offer one mechanism to register such objects. Since whiteboard pattern allows for simpler code than having to access a service to register with adding new API was dismissed.

8.2 Web Application Events

8.2.1 Limiting events

Instead of just sending web application events to all event listeners registered in the OSGi service registry it would be conceivable that listeners may register with a osgi.http.service.target service property which defines an LDAP filter to limit the Http Whiteboard Services sending events to the listener service.

I am not sure whether this would really be of use.

8.2.2 Event Admin Service

Servlet Events could be bridged into Event Admin Service events.

I am omitting such bridging right now because I am not sure of its use.
8.3 HTTP Sessions

The simplest implementation for HTTP Sessions would be to have a single HTTP Session backed by servlet container and thus shared amongst all Servlets and their servlet contexts. Yet, this would probably be unexpected for these applications which have separate servlet contexts and thus separate attribute value spaces but still share the same HTTP Session.

8.4 Resources

Alternatively to the proposed Resource servlet it might be conceivable to have the osgi.http.whiteboard.path and osgi.http.whiteboard.prefix properties on an Http Context service to register resources to be served through the given Http Context. In this case the path property must be a prefix pattern. If we support multi-value properties, the pattern and prefix properties must provide the same number of values and they are put together by the same index; i.e. path[0] → prefix[0], path[1] → prefix[1], etc.

While this solution looks appealing, I am not sure, whether there is a conceptual fit between the Http Context service and the resource registration. On the other hand resources are served (resolved actually) through an Http Context, so to register resources an Http Context is always required.

8.5 Deprecated HttpService

Instead of updating the Http Service it has been decided to create a new specification for the Http Whiteboard Service and leave the Http Service as is. The new specification is split up in different packages, one for all whiteboard related stuff, one for the new Http Service Runtime and one for the new ServletContextHelper.

9 Security Considerations

Bundles that need to register a servlet, listener, resource filter, or http context must be granted ServicePermission[Interface Name, REGISTER] where interface name is the whiteboard interface the service is registered for.

Bundles that need to iterate the servlets, listeners, resources, filters, or servlet context helpers registered with the system must be granted ServicePermission[interface name, GET] to retrieve the services from the service registry.

In addition if a whiteboard service wants to be associated with a shared servlet context helper registered by another bundle, the bundle registering the whiteboard service must be granted ServicePermission[org.osgi.service.http.context.ServletContextHelper, GET].

Bundles that need to introspect the state of the Http Whiteboard Service runtime will need PackagePermission[org.osgi.service.http.runtime, IMPORT] and ServicePermission[org.osgi.service.http.runtime.HttpServiceRuntime, GET] to obtain the HttpServiceRuntime service and access the DTO types.
10 Document Support

10.1 References

[4]. Portable Java SE/EE Contracts, RFC 180, work in progress

10.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Felix Meschber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Adobe Systems Incorporated</td>
</tr>
<tr>
<td>Address</td>
<td>Barfüsserplatz 6, 4055 Basel, Switzerland</td>
</tr>
<tr>
<td>Voice</td>
<td>+41 61 226 55 49</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:fmeschbe@adobe.com">fmeschbe@adobe.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Carsten Ziegeler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Adobe Systems Incorporated</td>
</tr>
<tr>
<td>Address</td>
<td>Barfüsserplatz 6, 4055 Basel, Switzerland</td>
</tr>
<tr>
<td>Voice</td>
<td>+41 61 226 55 0</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:cziegele@adobe.com">cziegele@adobe.com</a></td>
</tr>
</tbody>
</table>

10.3 Acronyms and Abbreviations

10.4 End of Document
RFC 190 - Declarative Services Enhancements

Draft
84 Pages

Abstract
Declarative Services provide functionality to implement Dependency Injection programming in OSGi based applications. This RFC proposes describing adding an additional management API, better support of property handling through annotation interfaces, new features introduced in Configuration Admin 1.4 and 1.5, service scopes, and defines an extender namespace.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

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
I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS Delineated Above.

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0 Document Information..................................................................................................................2
  0.1 License....................................................................................................................................2
  0.2 Trademarks...............................................................................................................................3
  0.3 Feedback....................................................................................................................................3
  0.4 Table of Contents......................................................................................................................3
  0.5 Terminology and Document Conventions..................................................................................4
  0.6 Revision History.........................................................................................................................4

1 Introduction....................................................................................................................................8

2 Application Domain.......................................................................................................................8
  2.1 Terminology + Abbreviations....................................................................................................9

3 Problem Description.....................................................................................................................9
  3.1 Management.............................................................................................................................9
  3.2 Bound Service Properties........................................................................................................9
  3.3 Lookup Strategy......................................................................................................................9
  3.4 Deployer wants a mandatory reference...................................................................................10

4 Requirements..............................................................................................................................10

5 Technical Solution......................................................................................................................10
  5.1 Introspective API....................................................................................................................10
  5.2 Event Method Signature..........................................................................................................11
  5.3 API version..............................................................................................................................12
  5.4 XML Descriptor Namespace..................................................................................................13
  5.5 Extender Capability................................................................................................................13
  5.6 Configuration Annotation Types............................................................................................13
    5.6.1 Custom annotations to define properties............................................................................13
    5.6.2 Mapping Annotation Elements to Component Properties.............................................14
    5.6.3 Mapping Component Properties to Annotation Elements............................................15
    5.6.4 Generation of property information in @Component annotation.....................................16
5.7 Integration with the Configuration Admin Service .................................................. 17
  5.7.1 Targeted PIDs ........................................................................................................ 17
  5.7.2 Multi-location binding ........................................................................................ 17
  5.7.3 Multiple PIDs ....................................................................................................... 17
  5.7.4 service.pid Property ............................................................................................ 18
5.8 Service Scopes ........................................................................................................... 18
  5.8.1 Providing Services .............................................................................................. 18
  5.8.2 Consuming Services ........................................................................................... 18
  5.8.3 Annotation Changes ............................................................................................ 19
  5.8.4 Schema Changes ................................................................................................. 19
5.9 Lookup References ..................................................................................................... 19
5.10 Minimum Cardinality Property ................................................................................ 19

6 Data Transfer Objects .................................................................................................. 20

7 Java API ....................................................................................................................... 20

8 Considered Alternatives ............................................................................................... 81
  8.1 Diagnostic API ......................................................................................................... 81
  8.2 @Component annotation inheritance ...................................................................... 81
  8.3 Create separate Service annotation (Bug 2140) ...................................................... 82
  8.4 Component provided service properties (Bug 2250) ............................................... 82
  8.5 Create separate Property annotation (Bug 2141) ...................................................... 82
  8.6 New Life Cycle States .............................................................................................. 82

9 Security Considerations ............................................................................................... 82

10 Document Support .................................................................................................... 83
  10.1 References ............................................................................................................. 83
  10.2 Author’s Address ................................................................................................... 83
  10.3 Acronyms and Abbreviations ............................................................................... 83
  10.4 End of Document .................................................................................................. 83

0.5 Terminology and Document Conventions
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>Sept. 17 2012</td>
<td>Initial version from RFP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Felix Meschberger, Adobe Systems Incorporated, <a href="mailto:fmeschbe@adobe.com">fmeschbe@adobe.com</a></td>
</tr>
<tr>
<td>Revision</td>
<td>Date</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 2<sup>nd</sup> | Sept. 24 2012 | Updates from Basel F2F:  
  - Integrate Administrative API and design it to be DTO-style  
  - Simplify security (ServicePermission [ServiceComponentRuntime, GET] is enough)  

Felix Meschberger, Adobe Systems Incorporated, fmeschbe@adobe.com |
| 3<sup>rd</sup> | 06/06/13   | Update from Orlando F2F and BJ's feedback on the CPEG mailing list  
  - Relable the administrative API as the diagnostic API  
  - fleshed out annotation inheritance but suggest to actually remove it (section 5.7.4, Supporting Inheritance)  
  - Added section 5.9, Service Scopes  

Felix Meschberger, Adobe Systems Incorporated, fmeschbe@adobe.com |
| 4<sup>th</sup> | 07/08/13   | Update from Palo Alto F2F  
  - Removed separate service annotations  
  - Removed annotation inheritance  
  - Removed setting properties through the component  
  - Updated DTOs  
  - New suggestion for property annotation  

Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| 5<sup>th</sup> | 07/19/13   | Update from CPEG Call (18/07/13)  
  - Removed alternative property annotation proposals  
  - Clarified annotation based approach  

Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| 6<sup>th</sup> | 09/18/13   | Update after BJ's review, major changes:  
  - Removed props attribute from @Component annotation and allow multiple annotation arguments in the lifecycle methods  
  - Remove DISPOSED state  

Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| 7<sup>th</sup> | 01/10/13   | Update with feedback from CPEG call:  
  - Add information (back) from annotations as properties to descriptor XML  

Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| 8<sup>th</sup> | 17/10/13 | Update from Bug 2515  
Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com |
| 9<sup>th</sup> | 24/10/13 | Update from CPEG call  
• Clarified annotation field to property mapping |
| 10<sup>th</sup> | 2013-11-11 | API/Javadoc improvements (up through section 5.6)  
BJ Hargrave, IBM |
| 11<sup>th</sup> | 2013-11-15 | Rewrite the Configuration Admin support changes. Added support for multiple PIDs.  
BJ Hargrave, IBM |
| 12<sup>th</sup> | 2013-11-19 | Improved the annotation element to property name conversion rules.  
Added '$' to annotation element to property name conversion rules.  
Some other text improvements to clarify meaning, Reference RFC 208  
AttachmentDefinition annotation for the property name for an annotation element.  
BJ Hargrave, IBM |
| 13<sup>th</sup> | 2013-11-20 | Updated with feedback from Peter Kriens. The signatures for event methods are expanded to allow more flexibility. The changes for prototype services are expanded to include the text from RFC 195.  
BJ Hargrave, IBM |
| 14<sup>th</sup> | 2013-11-22 | Added diagram for DTOs. Removed reference to id element of  
AttachmentDefinition annotation. Defined special string for component name that can be used in configurationPid(). New coercion table from property value to annotation element value.  
BJ Hargrave, IBM |
| 15<sup>th</sup> | 2013-12-06 | Reviewed changes in CPEG call and accepted them. Minor changes to coercion rules including when no property is present.  
BJ Hargrave, IBM |
| 16<sup>th</sup> | 2013-12-16 | Allow single Map argument for event methods. Add support for lookup strategy references. Added minimum cardinality property for references.  
BJ Hargrave, IBM |
| 17<sup>th</sup> | 2013-12-19 | Reviewed changes in CPEG call and accepted them.  
BJ Hargrave, IBM |
| 18<sup>th</sup> | 2014-01-10 | Updated targeted PID support to state that creation and deletion of Configurations with matching targeted PIDs are treated as a configuration modification.  
BJ Hargrave, IBM |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>19th</td>
<td>2014-01-24</td>
<td>The minimum cardinality property is to be supported for older component namespaces. Configuration property names starting with <code>&lt;refname&gt;</code>, are reserved for future use by the specification. The DTOs were reviewed to ensure there were no cycles. There were none, but there were duplicate instances of ReferenceDTO. BoundReferenceDTO was changed to use the reference name rather than a ReferenceDTO. BJ Hargrave, IBM</td>
</tr>
<tr>
<td>20th</td>
<td>2014-02-07</td>
<td>Updated figure in 5.1 to fix errors in cardinalities. Refactored dtos into a package name ending with dto in conformance with updated dto package naming rules. BJ Hargrave, IBM</td>
</tr>
<tr>
<td>21st</td>
<td>2014-04-22</td>
<td>Bug 2668: Address service types of Map, ServiceReference and ServiceObjects in selecting the event method signature. BJ Hargrave, IBM</td>
</tr>
<tr>
<td>22nd</td>
<td>2014-06-03</td>
<td>Bug 2679: Side effects of enable/disable component are asynchronous. BJ Hargrave, IBM</td>
</tr>
<tr>
<td>23rd</td>
<td>2014-06-07</td>
<td>Bug 2679: Return Promise for enable/disable component in ServiceComponentRuntime. This allows the caller to be notified when the actions that result from the enabled state change are completed. Bug 2692: Added a section to describe how service.pid component properties are not replaced but aggregated when there are multiple property sources. BJ Hargrave, IBM</td>
</tr>
<tr>
<td>24th</td>
<td>2014-06-25</td>
<td>Update DTOs to split BoundReferenceDTO into SatisfiedReferenceDTO and UnsatisfiedReferenceDTO providing a DTO for both satisfied and unsatisfied references. Distinguish between component configurations which are unsatisfied because of a missing required configuration or unsatisfied references. Provide access to DTOs for unsatisfied references. BJ Hargrave, IBM</td>
</tr>
<tr>
<td>25th</td>
<td>2014-06-28</td>
<td>Add <code>targetServices</code> to UnsatisfiedReferenceDTO. BJ Hargrave, IBM</td>
</tr>
</tbody>
</table>
1 Introduction

This Declarative Services Enhancements RFC defines functionality currently implemented in some implementations of the specification or currently requiring special component code as part of the OSGi Declarative Services Specification.

2 Application Domain

Declarative Services (chapter 121 in the OSGi specifications) defines a POJO programming model for OSGi services. This model requires Service Component class be implemented in a certain way and the XML component descriptions be authored.

A number of changes have occurred over the past few years that affect DS:

- Apache Felix has implemented this specification and added a very popular management API. This same API was subsequently also used by Eclipse. The API is registered as an Admin service and provides methods to query the state of the components.

- Configuration Admin introduced a number of changes affecting DS.
  - Targeted PIDs — Targeted PIDs were introduced in Configuration Admin 1.5. These are PIDs that can distinguish between bundles and therefore target the configuration for a specific bundle. (See 104.3.2). Targeted PIDs allows for the PID to be further constrained by the bundle symbolic name, bundle version and bundle location. For ManagedService and ManagedServiceFactory services, Configuration Admin will always supply the configuration with the most targeted PID. Section 104.3.3, Extenders and Targeted PIDs, requires extenders such as SCR to properly support Targeted PIDs.
  - Multiple PIDs — Multiple PID support was introduced in Configuration Admin 1.4. Multiple PID support allows a configuration target to receive configurations for multiple PIDs. This is of more value with multi-location binding since a common configuration can be shared among a set of bundles.
  - Multi-location binding — Multi-location binding was introduced in Configuration Admin 1.4. Multi-location binding allows a configuration to avoid binding to a specific bundle location by prefixing the location with “?”. This way multiple bundles can share the same configuration. Java permissions can be used to limit a bundle’s access to a configuration.

- RFC 195, Service Scopes, defines a new mechanism to access services from the service registry. This mechanism allows to get new service instances on demand instead of either always the same instance globally (regular service) or per bundle (service factory).
DS supports Reference annotations that can be applied to *bind* methods. The reference name and service type can be inferred from the annotated method signature. A reference element in the component description is generated which references the annotated bind method.

DS supports four reference cardinality modes. In addition to supporting more than one reference, a reference can be optional or mandatory. That is, a reference can be satisfied with zero or one bound service.

### 2.1 Terminology + Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS</td>
<td>Declarative Services</td>
</tr>
<tr>
<td>POJO</td>
<td>Plain old Java Object; term use for objects not implementing and framework specific plumbing such as Servlet API, Spring API, or OSGi API.</td>
</tr>
<tr>
<td>SCR</td>
<td>Service Components Runtime; generally the implementation of the Declarative Services Specification; also the name of the Apache Felix implementation (Apache Felix SCR).</td>
</tr>
</tbody>
</table>

### 3 Problem Description

#### 3.1 Management

There is no official API yet to introspect and thus manage the declared service components. To work around this missing functionality the Apache Felix project defined such an API which is also implemented by current versions of the Eclipse Equinox implementation.

This current API has some short-comings which are addressed by a new proposal.

#### 3.2 Bound Service Properties

As of DS Version 1.1 the service registration properties of bound services can be provided to the components using an optional `java.lang.Map` argument. While this allows for great capability introspecting the bound service it lacks support for the natural ordering defined for `org.osgi.framework.ServiceReference` which implements `Comparable`.

The solution applied today is to either use the greedy service binding policy option as defined in DS Version 1.2 or to implement such ordering in the component itself. Such implementation, though, is pure template code and thus error prone load to developers.

#### 3.3 Lookup Strategy

Reference annotations can only be applied to methods and so only really support the event strategy of referenced service access. To use only lookup strategy for a reference, you either cannot use DS annotations or you must supply an otherwise unused bind method to be annotated by the Reference annotation.
3.4 Deployer wants a mandatory reference

A developer can author a component to allow for an optional reference. The component can also be authored to support more than one reference. That is, the lower bound is 0 and the upper bound is 1 or n. However, a deployer may want to narrow the cardinality of the reference to require more than 0 bound services.

4 Requirements

R-1 The solution MUST define an API to introspect declared components and their configurations.

R-2 The solution MUST make it possible to leverage the natural ordering capability of ServiceReference along with the service instance provisioning through the event method by allowing the new signature:

   void <method-name>(<parameter-type>, ServiceReference);

R-3 The solution MUST define a new way to define component properties by referencing a java annotation class. The fields of this annotation class are used to define properties for the component.

R-4 The solution MUST define the osgi.extender capability for DS in accordance with the core specification rules for the osgi.extender name space.

R-5 The solution MUST support targeted PIDs according to Configuration Admin 1.5.

R-6 The solution MUST support multi-location binding from Configuration Admin 1.4

R-7 The solution MUST support the multiple PIDs feature from Configuration Admin 1.4

R-8 The solution MUST support the Service scopes introduced in the core.

R-9 The solution MUST allow references to be declared in annotations without requiring a bind method.

R-10 The solution MUST allow a deployer to raise the minimum number of bound services for a reference to be considered satisfied.

5 Technical Solution
5.1 Introspective API

The introspective API is structured after the component descriptor within its own package org.osgi.service.component.runtime. The ServiceComponentRuntime service interface is the API entry point. It is registered by the SCR and provides access to properties of the implementation and to the properly declared components. Any components whose descriptor cannot be validated is considered unknown and thus is not available through the ServiceComponentRuntime service.

Each component declaration is accessible through the ServiceComponentRuntime as an instance of the ComponentDescriptionDTO class. The ComponentDescriptionDTO provides access to the static declaration.

Components actually are available from the ServiceComponentRuntime as ComponentConfigurationDTO instances. Each ComponentConfigurationDTO links back to its declaring ComponentDescriptionDTO.
Since a single component declaration may be associated with multiple component configurations – for example due to multiple Configurations from ConfigurationAdmin – a single ComponentDescriptionDTO instance may be referred to by multiple ComponentConfigurationDTO instances.

To address the similar distinction between the declaration of references and references of a component configuration, the ComponentDescriptionDTO object provides the declared references as ReferenceDTO objects while the ComponentConfigurationDTO returns SatisfiedReferenceDTO and UnsatisfiedReferenceDTO object representing the references of the component configuration which may be satisfied or unsatisfied.

To simplify remote management, the ComponentDescriptionDTO, ComponentConfigurationDTO, ReferenceDTO, SatisfiedReferenceDTO, and UnsatisfiedReferenceDTO types are defined as DTO-style classes and integrate with the API defined by Data Transfer Objects [3].

A bundle wishing to access the DTOs must have ServicePermission[ServiceComponentRuntime, GET] to get the ServiceComponentRuntime service.

### 5.2 Event Method Signature

The signatures for event strategy methods in Section 112.3.2, Event Methods are modified to allow for an new parameter, ServiceObjects and also to allow for additional flexibility in the ordering and number of parameters. Section 112.3.2 is updated to state:

The prototype of the events methods is

```java
void <method-name>(<arguments>);
```

The event methods can take one or more arguments. Each argument must be of one of the following types:

- `<service-type>` – This is the bound service object.
- `ServiceReference` – This is the ServiceReference for the bound service.
- `ServiceObjects` – This is the ServiceObjects for the bound service. This argument type can only be used when the reference is `scope=prototype`.
- `Map` – This is a map of the service properties for the bound service.

A suitable method is selected using the following priority:

1. The method takes a single argument and the type of the argument is `org.osgi.framework.ServiceReference`. This method will receive the ServiceReference for the bound service.
2. The method takes a single argument and the type of the argument is `org.osgi.framework.ServiceObjects`. This signature can only be used when the reference is `scope=prototype`. This method will receive the ServiceObjects for the bound service.
3. The method takes a single argument and the type of the argument is the type specified by the reference's interface attribute. This method will receive the bound service.
4. The method takes a single argument and the type of the argument is assignable from the type specified by the reference’s interface attribute. If multiple methods match this rule, this implies the method name is overloaded and SCR may choose any of the methods to call. This method will receive the bound service.

5. The method takes a single argument and the type of the argument is java.util.Map. This method will receive the service properties for the bound service.

6. The method takes two or more arguments and the type of each argument must be the type specified by the reference’s interface attribute, assignable from the type specified by the reference’s interface attribute, org.osgi.framework.ServiceReference, org.osgi.framework.ServiceObjects or java.util.Map. If multiple methods match this rule, this implies the method name is overloaded and SCR may choose any of the methods to call. In the case where the type of the bound service is Map, ServiceReference or ServiceObjects, the first argument of that type will receive the bound service. If the selected method has more than one argument of that type, the remaining arguments of that type will receive the service properties Map, ServiceReference or ServiceObjects, respectively, for the bound service.

The above changes allow for more flexibility in the signatures of event methods, similar to the lifecycle methods, including multiple arguments of different type in any order.

The additional signatures are only supported if the component is declared in a descriptor with namespace http://www.osgi.org/xmlns/scr/v1.3.0 or newer.

5.3 API version
The DS API is exported as version 1.3 to reflect these updates.

5.4 XML Descriptor Namespace
The XML descriptor namespace is changed to

http://www.osgi.org/xmlns/scr/v1.3.0

New functionality defined in this specification requires component to be registered with this namespace. Otherwise, for backwards compatibility reasons, neither the added event method signature nor the new prototype service scope an be used.

5.5 Extender Capability
The DS implementation bundle must declare the following extender capability:

```
Provide-Capability: osgi.extender;
    osgi.extender="osgi.component";
    uses:="org.osgi.service.component";
    version:Version="1.3"
```

5.6 Configuration Annotation Types

5.6.1 Custom annotations to define properties
Component properties can be defined through custom annotation types containing the property names, property types and default values. For example:

```
@interface Config {
    boolean enabled() default true;
    String[] names() default ["a", "b"];
    String topic() default MyComponent.DEFAULT_TOPIC_PREFIX + "/topic";
}
```
The lifecycle methods for activation, deactivation and modification can use a signature which allows them to specify one or more annotation types as arguments. The annotation elements with default values from all the annotation types used in the lifecycle method arguments are added via property elements to the generated component descriptions. The order of processing is: first annotation types used as arguments to the activate method, followed by annotation types used as arguments to from the modified method and finally annotation types used as arguments to the deactivate method. If a lifecycle method has more than one annotation type arguments, these are processed in the order of the method arguments.

New signatures for the lifecycle methods are supported, taking one or more arguments of annotation types. These are additional possible arguments. The following details the search for a lifecycle method:

1. The method takes a single argument and the type of the argument is org.osgi.service.component.ComponentContext.
2. The method takes a single argument and the type of the argument is org.osgi.framework.BundleContext.
3. The method takes a single argument and the type of the argument is an annotation type.
4. The method takes a single argument and the type of the argument is the java.util.Map.
5. For deactivation methods only: The method takes a single argument and the type of the argument is the int.
6. For deactivation methods only: The method takes a single argument and the type of the argument is the java.lang.Integer.
7. The method takes two or more arguments and the type of each argument must be org.osgi.service.component.ComponentContext, org.osgi.framework.BundleContext, java.util.Map or an annotation type. For the deactivation method int or java.lang.Integer are allowed types as well. If multiple methods match this rule, this implies the method name is overloaded and SCR may choose any of the methods to call.
8. The method takes zero arguments.

5.6.2 Mapping Annotation Elements to Component Properties

Each element of an annotation type used as a lifecycle method argument is mapped to a component property. The property name is derived from the element name. Certain common property name characters, such as "." are not valid in Java identifiers. So the name of an annotation element is converted to the property name as follows:

- A single dollar sign ('$') is removed unless it is followed by another dollar sign in which case the two consecutive dollar signs ('$$') are converted to a single dollar sign.
• A single underscore (\_) is converted into a dot (\.) unless is it followed by another underscore in which case the two consecutive underscores (\__) are converted to a single underscore.

• All other characters and numbers are unchanged.

The processing starts at the beginning of the name. Examples:

<table>
<thead>
<tr>
<th>Annotation Element Name</th>
<th>Component Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>myProperty143</td>
<td>myProperty143</td>
</tr>
<tr>
<td>$new</td>
<td>new</td>
</tr>
<tr>
<td>my$prop</td>
<td>my$prop</td>
</tr>
<tr>
<td>_secret</td>
<td>.secret</td>
</tr>
<tr>
<td>another__prop</td>
<td>another_prop</td>
</tr>
<tr>
<td>three___prop</td>
<td>three__prop</td>
</tr>
<tr>
<td>four_$__prop</td>
<td>four_.prop</td>
</tr>
<tr>
<td>five_$_prop</td>
<td>five..prop</td>
</tr>
</tbody>
</table>

The property type can directly be derived from the type of the annotation element. All types supported for annotation elements can be used except for annotation types.

If the annotation element type is Class or Class[], the corresponding property type is String or String[] whose value comes from the `Class.getName()` method.

If the annotation element type is an enumeration type or an array thereof, the corresponding property type is String or String[] whose value comes from the `Enum.name()` method.

Annotation elements with an annotation type or array thereof are not supported. A tool processing the DS annotations should declare an error during processing in this case. Any default value is ignored.

If an annotation element has a default value, the tool processing the DS annotation must add a property element to the component description with the property name converted from the annotation element name and the property value from the annotation element's default value. Default values of type Class or an enumeration type are mapped to String values in the component description using `Class.getName()` and `Enum.name()` respectively. If an annotation element does not have a default value, the tool processing the DS annotation must not add a property element to the component description for the converted property name.

### 5.6.3 Mapping Component Properties to Annotation Elements

If an annotation type is used as an argument for a lifecycle method, SCR must create an object implementing the annotation type and maps the available component properties to the annotation elements. The name of the annotation element is converted to the property name as described above. The property value might need to be coerced to the type of the annotation element. In the following coercion table, the columns are source types (for example, component property type) and the rows are target types (for example, annotation element type). The property value is v, `number` is the primitive numerical type and `Number` is the wrapper numerical type. An invalid coercion is represented by throw. Such a coercion attempt must result in throwing a ComponentException. Any other coercion error, such as parsing a non-numerical String to a number or inability to coerce a string into a Class or enum object, must be wrapped in a ComponentException and thrown.
<table>
<thead>
<tr>
<th>source</th>
<th>target</th>
<th>String</th>
<th>Boolean</th>
<th>Character</th>
<th>Number</th>
<th>Collection/array</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>v</td>
<td>v.toString()</td>
<td>v.toString()</td>
<td>v.toString()</td>
<td>If v has no elements, null; otherwise the first element of v is coerced.</td>
<td></td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean.parseBoolean(v)</td>
<td>v.toString()</td>
<td>v.toString()</td>
<td>v.toString()</td>
<td>If v has no elements, false; otherwise the first element of v is coerced.</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>v.length() &gt; 0 ? v.toString() : 0</td>
<td>v.toString()</td>
<td>v.toString()</td>
<td>v.toString()</td>
<td>If v has no elements, false; otherwise the first element of v is coerced.</td>
<td></td>
</tr>
<tr>
<td>number</td>
<td>Number.parseNumber(v)</td>
<td>v.toString()</td>
<td>v.toString()</td>
<td>v.toString()</td>
<td>If v has no elements, false; otherwise the first element of v is coerced.</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>Bundle.loadClass(v)</td>
<td>throw</td>
<td>throw</td>
<td>throw</td>
<td>If v has no elements, null; otherwise the first element of v is coerced.</td>
<td></td>
</tr>
<tr>
<td>EnumType</td>
<td>EnumType.valueOf(v)</td>
<td>throw</td>
<td>throw</td>
<td>throw</td>
<td>An array the size of v is created and each element of v is coerced into it.</td>
<td></td>
</tr>
<tr>
<td>annotation</td>
<td>throw</td>
<td>throw</td>
<td>throw</td>
<td>throw</td>
<td>An array the size of v is created and each element of v is coerced into it.</td>
<td></td>
</tr>
</tbody>
</table>

Component properties whose names do not map to annotation elements are ignored. If the implementation needs to access these additional properties, it can use a method signature which also receives the properties map in addition to the annotation type.

If there is no corresponding component property for an annotation element, the returned value must be:

- 0 for numerical types and char
- false for boolean
- null for String, Class, enum and arrays

For elements of an annotation type, a ComponentException must be thrown when calling the element.

This new lifecycle method signatures supporting annotation type arguments are only supported when the component description is declared with namespace http://www.osgi.org/xmlns/scr/v1.3.0 (or later).

### 5.6.4 Generation of property information in @Component annotation

The @Component annotation contains two different approaches to define component property values through two different elements: property and properties. In addition, annotations types can now be used to define component property values as well. As annotation elements are unordered when the DS annotations are processed by a tool, the current specification does not define the order of the corresponding elements in the generated component description. However the order in the generated component description defines the order of processing and later property values with the same property name override previously declared ones.

This should be clarified by defining an order of output to the generated component description:

1. Properties defined through annotation types used in the signature of the lifecycle methods.
   1. Annotations types used in the activate method in the order of parameters in the signature
   2. Annotations types used in the modified method in the order of parameters in the signature
   3. Annotations types used in the deactivate method in the order of parameters in the signature
2. property element from @Component

3. properties element from @Component

This means that the properties defined through annotation types are declared first in the generated component description, followed by all properties defined through the @Component.property element and finally the properties entries defined through the @Component.properties element.

Since property values defined later in the component description override property values defined earlier in the component description, this means that property values defined in @Component.properties override property values defined in @Component.property which override values defined by default values in annotation types used in lifecycle method signatures.

5.7 Integration with the Configuration Admin Service

DS integrates with the Configuration Admin Service. Therefore SCR must support the latest additions to the Configuration Admin Service.

5.7.1 Targeted PIDs

Section 112.7 Deployment must be updated to refer to 104.3.3 in the Configuration Admin specification and to state that SCR must use the most targeted PID to select the matching Configuration for a component. SCR must also handle the creation and deletion of Configurations with matching targeted PIDs such that:

- The deletion of an in-use Configuration will act as a configuration modification if there is another Configuration with a matching targeted PID.

- The creation of a Configuration with a more precise matching targeted PID will act as a configuration modification.

5.7.2 Multi-location binding

When Java permissions are enabled, SCR must make sure these bindings are properly supported. If the multi-location is just the question mark, no additional checks must be made, as this configuration can be used by any bundle. If a region is specified, SCR must check whether the component bundle has the required permission as outlined in Configuration Admin, Section 104.7.6.

5.7.3 Multiple PIDs

DS must be updated to allow a component to specify multiple configuration PIDs for a component. The configuration-pid attribute of the component element will be updated to allow multiple pid values to be specified using a whitespace separated list. For example:

```xml
<component configuration-pid="com.acme.host com.acme.system" ...>
```

The configurationPid element of the @Component annotation is updated to support multiple values.

```java
String[] configurationPid() default {"$"};
```

This is source compatible for existing code using the @Component annotation even though the type of the element is changed from String to String[]. A special pid string, "$", can be used with configurationPid() to represent the actual name of the component. The tool processing the annotation into a component description must replace the special string with the actual component name.

When multiple PIDs are specified for a component description, SCR must aggregate the configuration properties from the matching Configurations in the order the PIDs are specified such that properties in Configurations for later specified PIDs take precedence over properties in Configurations for earlier specified PIDs. The creation,
updating and deletion for a matching Configuration for any of the PIDs, is a configuration change under section 112.7.1 of the spec and can result in a component configuration becoming satisfied, unsatisfied or modified.

If multiple PIDs are used for a component, only one of the PIDs can be for a factory configuration. If more than one of the PIDs are for factory configurations, this is a configuration error which SCR must log. If this configuration error occurs and the configuration-policy is optional, the component configuration can be satisfied without any of the offending Configurations. If this configuration error occurs and the configuration-policy is required, the component configuration is unsatisfied.

Sections 112.6 and 112.7 must be updated to reflect support for multiple PIDs.

5.7.4 service.pid Property

When aggregating properties for the component properties, normally properties from higher precedence sources replace properties from lower precedence sources (see 112.6 in the DS specification). The service.pid property will not have this behavior and instead SCR must aggregate the service.pid values from all sources into a Collection<String> values. This collection must have an iteration order such that the first item is the service.pid value from the lowest precedence source and the last item is the service.pid value from the highest precedence source. For multiple configuration in 5.7.3, the order of the service.pid values is the order the PIDs are specified in the component description.

The values in the Collection<String> are the values as they come from the source which, for configurations, may be different than the PID value(s) in the component description.

5.8 Service Scopes

RFC 195, Service Scopes, defines a new mechanism to access services from the service registry. This mechanism allows to get new service instances on demand instead of either always the same instance globally (regular service) or per bundle (service factory).

The introduction of prototype scope services by RFC 195 means we also need to update DS to support this new service feature.

5.8.1 Providing Services

The servicefactory attribute on the service element is deprecated and replaced by a scope attribute supporting the values: singleton (default), bundle and prototype. servicefactory=false maps to scope=singleton and servicefactory=true maps to scope=bundle.

This allows SCR to support components being prototype scope services. Since DS never registers the actual component object (that is, even for scope=singleton, DS always registers a ServiceFactory to delay component creation and activation), components will never be visible in the service registry with service.scope=singleton.

5.8.2 Consuming Services

A scope attribute is added to the reference element. The scope attribute supports the values: bundle (default) and prototype. When using scope=bundle, all references to the service by components in the same bundle will share the same service object. That is, SCR must use BundleContext.getService to obtain the service object. When using scope=prototype, each instance of the component will use a different instance of the service. That is, SCR must use BundleContext.getServiceObjects to obtain the service object and the referenced service must have service.scope=prototype. A service without service.scope=prototype cannot be used as a bound service for a scope=prototype reference since it cannot fulfill the requirement to create multiple service instances for the bundle.
The valid signatures for bind, updated and unbind will be extended to allow ServiceObjects to be injected. For example:

```java
void bind(ServiceObjects);
```

See 5.2 Event Method Signature.

### 5.8.3 Annotation Changes

The DS Annotations will also be updated to support these new features.

**Enum** `ServiceScope` is added with values `SINGLETON`, `BUNDLE` and `PROTOTYPE`.

```
ServiceScope scope() is added to the Component annotation. Component.servicefactory() is deprecated and ignored when Component.scope() is specified.
```

**Enum** `ReferenceScope` is added with values `BUNDLE` and `PROTOTYPE`.

```
ReferenceScope scope() is added to the Reference annotation.
```

### 5.8.4 Schema Changes

The DS XML Schema is updated to add a `scope` attribute to the `service` and the `reference` elements. The `servicefactory` attribute of the `service` element is removed since it is replaced by the new `scope` attribute.

### 5.9 Lookup References

A new LookupReference annotation is defined and a reference element is added to the Component annotation. This allows the use of DS annotation for lookup strategy access to referenced services. The reference element can specify a number of LookupReference annotations from which reference elements can be generated in the component description. The LookupReference annotation is very similar to the existing Reference element, but requires the name and service elements to be specified since the LookupReference annotation is not applied to a bind method from whose signature the reference name and service type can be inferred. The LookupReference reference annotation also does not contain updated or unbind elements.

### 5.10 Minimum Cardinality Property

A new component property is defined called the `minimum cardinality property` of a reference. Like the target property described in 112.6, the minimum cardinality property name is prefixed by the reference name to identify to which reference the minimum cardinality property applies. So the minimum cardinality property name takes the form:

```
<refname>.cardinality.minimum
```

where `<refname>` is the name of the reference.

The minimum cardinality property can be used by a deployer (e.g. via Configuration Admin) to raise the floor of the cardinality not to exceed the ceiling. That is, a 0..1 cardinality can be raised to a 1..1 cardinality with a minimum cardinality property value of 1. And a 0..n or 1..n cardinality can be raised to a m..n cardinality with a minimum cardinality property value m which must be greater than 0. A 1..1 or 1..n cardinality cannot lowered to a 0..1 or 0..n cardinality since the component developer will have coded to expect at least one bound service. Therefore the minimum cardinality property value must be coercible (see the coercion table above) to a positive integer. If a reference has a minimum cardinality property and its value cannot be coerced into a positive integer, the reference's minimum cardinality property must be ignored. For references with declared cardinalities of 0..1
and 1..1, the only valid value for the reference's minimum cardinality property is 1. Any other value must be ignored.

The minimum cardinality property must be supported for component in older namespaces.

Also, all component property names starting with the reference name following by a period, <refname>., are reserved for use by the specification.

6 Data Transfer Objects

The ServiceComponentRuntime service allows for the programmatic enablement and disablement of components as well as access to the state of components and component configurations. In particular the service provides these methods:

- ComponentDescriptionDTO getComponentDescriptionDTO(Bundle, String)
- Collection<ComponentDescriptionDTO> getComponentDescriptionDTOs(Bundle...)
- Collection<ComponentConfigurationDTO> getComponentConfigurationDTOs(ComponentDescriptionDTO)

See the JavaDoc for details.

7 Java API
<table>
<thead>
<tr>
<th>Package Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.service.component</td>
<td>22</td>
</tr>
<tr>
<td>Service Component Package Version 1.3.</td>
<td></td>
</tr>
<tr>
<td>org.osgi.service.component.annotations</td>
<td>34</td>
</tr>
<tr>
<td>Service Component Annotations Package Version 1.3.</td>
<td></td>
</tr>
<tr>
<td>org.osgi.service.component.runtime</td>
<td>63</td>
</tr>
<tr>
<td>Service Component Runtime Package Version 1.3.</td>
<td></td>
</tr>
<tr>
<td>org.osgi.service.component.runtime.dto</td>
<td>67</td>
</tr>
<tr>
<td>Service Component Runtime Data Transfer Objects Package Version 1.3.</td>
<td></td>
</tr>
</tbody>
</table>
Package org.osgi.service.component

@org.osgi.annotation.versioning.Version(value="1.3")

Service Component Package Version 1.3.

See: Description

### Interface Summary

<table>
<thead>
<tr>
<th>ComponentConstants</th>
<th>Defines standard names for Service Component constants.</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComponentContext</td>
<td>A Component Context object is used by a component instance to interact with its execution context including locating services by reference name.</td>
<td>26</td>
</tr>
<tr>
<td>ComponentFactory</td>
<td>When a component is declared with the factory attribute on its component element, the Service Component Runtime will register a Component Factory service to allow new component configurations to be created and activated rather than automatically creating and activating component configuration as necessary.</td>
<td>32</td>
</tr>
<tr>
<td>ComponentInstance</td>
<td>A ComponentInstance encapsulates a component instance of an activated component configuration.</td>
<td>33</td>
</tr>
</tbody>
</table>

### Exception Summary

<table>
<thead>
<tr>
<th>ComponentException</th>
<th>Unchecked exception which may be thrown by the Service Component Runtime.</th>
<th>Page</th>
</tr>
</thead>
</table>

Package org.osgi.service.component Description

Service Component Package Version 1.3.

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.component; version="[1.3,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.component; version="[1.3,1.4)"
```
### Interface ComponentConstants

**org.osgi.service.component**

@org.osgi.annotation.versioning.ProviderType

```java
public interface ComponentConstants
```

Defines standard names for Service Component constants.

#### Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPONENT_FACTORY</td>
<td>String</td>
<td>A service registration property for a Component Factory that contains the value of the factory attribute.</td>
<td>24</td>
</tr>
<tr>
<td>COMPONENT_ID</td>
<td>String</td>
<td>A component property that contains the generated id for a component configuration.</td>
<td>24</td>
</tr>
<tr>
<td>COMPONENT_NAME</td>
<td>String</td>
<td>A component property for a component configuration that contains the name of the component as specified in the name attribute of the component element.</td>
<td>23</td>
</tr>
<tr>
<td>DEACTIVATION_REASON_BUNDLE_STOPPED</td>
<td>int</td>
<td>The component configuration was deactivated because the bundle was stopped.</td>
<td>25</td>
</tr>
<tr>
<td>DEACTIVATION_REASON_CONFIGURATION_DELETED</td>
<td>int</td>
<td>The component configuration was deactivated because its configuration was deleted.</td>
<td>25</td>
</tr>
<tr>
<td>DEACTIVATION_REASON_CONFIGURATION_MODIFIED</td>
<td>int</td>
<td>The component configuration was deactivated because its configuration was changed.</td>
<td>25</td>
</tr>
<tr>
<td>DEACTIVATION_REASON_DISABLED</td>
<td>int</td>
<td>The component configuration was deactivated because the component was disabled.</td>
<td>24</td>
</tr>
<tr>
<td>DEACTIVATION_REASON_DISPOSED</td>
<td>int</td>
<td>The component configuration was deactivated because the component was disposed.</td>
<td>25</td>
</tr>
<tr>
<td>DEACTIVATION_REASON_REFERENCE</td>
<td>int</td>
<td>The component configuration was deactivated because a reference became unsatisfied.</td>
<td>24</td>
</tr>
<tr>
<td>DEACTIVATION_REASON_UNSPECIFIED</td>
<td>int</td>
<td>The reason the component configuration was deactivated is unspecified.</td>
<td>24</td>
</tr>
<tr>
<td>REFERENCE_TARGET_SUFFIX</td>
<td>String</td>
<td>The suffix for reference target properties.</td>
<td>24</td>
</tr>
<tr>
<td>SERVICE_COMPONENT</td>
<td>String</td>
<td>Manifest header specifying the XML documents within a bundle that contain the bundle's Service Component descriptions.</td>
<td>23</td>
</tr>
</tbody>
</table>

#### Field Detail

**SERVICE_COMPONENT**

```java
public static final String SERVICE_COMPONENT = "Service-Component"
```

Manifest header specifying the XML documents within a bundle that contain the bundle's Service Component descriptions.

The attribute value may be retrieved from the `Dictionary` object returned by the `Bundle.getHeaders` method.

**COMPONENT_NAME**

```java
public static final String COMPONENT_NAME = "component.name"
```
A component property for a component configuration that contains the name of the component as specified in the `name` attribute of the `component` element. The value of this property must be of type `String`.

**COMPONENT_ID**

```java
public static final String COMPONENT_ID = "component.id"
```

A component property that contains the generated id for a component configuration. The value of this property must be of type `Long`.

The value of this property is assigned by the Service Component Runtime when a component configuration is created. The Service Component Runtime assigns a unique value that is larger than all previously assigned values since the Service Component Runtime was started. These values are NOT persistent across restarts of the Service Component Runtime.

**COMPONENT_FACTORY**

```java
public static final String COMPONENT_FACTORY = "component.factory"
```

A service registration property for a Component Factory that contains the value of the `factory` attribute. The value of this property must be of type `String`.

**REFERENCE_TARGET_SUFFIX**

```java
public static final String REFERENCE_TARGET_SUFFIX = ".target"
```

The suffix for reference target properties. These properties contain the filter to select the target services for a reference. The value of this property must be of type `String`.

**DEACTIVATION_REASON_UNSPECIFIED**

```java
public static final int DEACTIVATION_REASON_UNSPECIFIED = 0
```

The reason the component configuration was deactivated is unspecified.

Since:

1.1

**DEACTIVATION_REASON_DISABLED**

```java
public static final int DEACTIVATION_REASON_DISABLED = 1
```

The component configuration was deactivated because the component was disabled.

Since:

1.1

**DEACTIVATION_REASON_REFERENCE**

```java
public static final int DEACTIVATION_REASON_REFERENCE = 2
```

The component configuration was deactivated because a reference became unsatisfied.
DEACTIVATION_REASON_CONFIGURATION_MODIFIED
public static final int DEACTIVATION_REASON_CONFIGURATION_MODIFIED = 3

The component configuration was deactivated because its configuration was changed.

Since: 1.1

DEACTIVATION_REASON_CONFIGURATION_DELETED
public static final int DEACTIVATION_REASON_CONFIGURATION_DELETED = 4

The component configuration was deactivated because its configuration was deleted.

Since: 1.1

DEACTIVATION_REASON_DISPOSED
public static final int DEACTIVATION_REASON_DISPOSED = 5

The component configuration was deactivated because the component was disposed.

Since: 1.1

DEACTIVATION_REASON_BUNDLE_STOPPED
public static final int DEACTIVATION_REASON_BUNDLE_STOPPED = 6

The component configuration was deactivated because the bundle was stopped.

Since: 1.1
A Component Context object is used by a component instance to interact with its execution context including locating services by reference name. Each component instance has a unique Component Context.

A component instance may have an activate method. If a component instance has a suitable and accessible activate method, this method will be called when a component configuration is activated. If the activate method takes a ComponentContext argument, it will be passed the component instance's Component Context object. If the activate method takes a BundleContext argument, it will be passed the component instance's Bundle Context object. If the activate method takes a Map argument, it will be passed an unmodifiable Map containing the component properties.

A component instance may have a deactivate method. If a component instance has a suitable and accessible deactivate method, this method will be called when the component configuration is deactivated. If the deactivate method takes a ComponentContext argument, it will be passed the component instance's Component Context object. If the deactivate method takes a BundleContext argument, it will be passed the component instance's Bundle Context object. If the deactivate method takes a Map argument, it will be passed an unmodifiable Map containing the component properties. If the deactivate method takes an int or Integer argument, it will be passed the reason code for the component instance’s deactivation.

ThreadSafe

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>void disableComponent(String name)</td>
<td>29</td>
</tr>
<tr>
<td>Enables the specified component name.</td>
<td></td>
</tr>
<tr>
<td>void enableComponent(String name)</td>
<td>28</td>
</tr>
<tr>
<td>Enables the specified component name.</td>
<td></td>
</tr>
<tr>
<td>org.osgi.framework.BundleContext getBundleContext()</td>
<td>28</td>
</tr>
<tr>
<td>Returns the BundleContext of the bundle which contains this component.</td>
<td></td>
</tr>
<tr>
<td>ComponentInstance GetComponentInstance()</td>
<td>28</td>
</tr>
<tr>
<td>Returns the Component Instance object for the component instance associated with this Component Context.</td>
<td></td>
</tr>
<tr>
<td>Dictionary&lt;String, Object&gt; getProperties()</td>
<td>27</td>
</tr>
<tr>
<td>Returns the component properties for this Component Context.</td>
<td></td>
</tr>
<tr>
<td>org.osgi.framework.ServiceReference&lt;?&gt; getServiceReference()</td>
<td>29</td>
</tr>
<tr>
<td>If the component instance is registered as a service using the service element, then this method returns the service reference of the service provided by this component instance.</td>
<td></td>
</tr>
<tr>
<td>org.osgi.framework.Bundle getUsingBundle()</td>
<td>28</td>
</tr>
<tr>
<td>If the component instance is registered as a service using the servicescope=&quot;bundle&quot; or servicescope=&quot;prototype&quot; attribute, then this method returns the bundle using the service provided by the component instance.</td>
<td></td>
</tr>
<tr>
<td>Object locateService(String name)</td>
<td>27</td>
</tr>
<tr>
<td>Returns the service object for the specified reference name.</td>
<td></td>
</tr>
<tr>
<td>Object locateService(String name, org.osgi.framework.ServiceReference&lt;?&gt; reference)</td>
<td>27</td>
</tr>
<tr>
<td>Returns the service object for the specified reference name and ServiceReference.</td>
<td></td>
</tr>
<tr>
<td>Object[] locateServices(String name)</td>
<td>27</td>
</tr>
<tr>
<td>Returns the service objects for the specified reference name.</td>
<td></td>
</tr>
</tbody>
</table>
Method Detail

getProperties

Dictionary<String,Object> getProperties()

Returns the component properties for this Component Context.

Returns:
The properties for this Component Context. The Dictionary is read only and cannot be modified.

locateService

Object locateService(String name)

Returns the service object for the specified reference name.

If the cardinality of the reference is 0..n or 1..n and multiple services are bound to the reference, the service with the highest ranking (as specified in its Constants.SERVICE_RANKING property) is returned. If there is a tie in ranking, the service with the lowest service id (as specified in its Constants.SERVICE_ID property); that is, the service that was registered first is returned.

Parameters:
name - The name of a reference as specified in a reference element in this component's description.

Returns:
A service object for the referenced service or null if the reference cardinality is 0..1 or 0..n and no bound service is available.

Throws: ComponentException - If the Service Component Runtime catches an exception while activating the bound service.

locateService

Object locateService(String name, org.osgi.framework.ServiceReference<?> reference)

Returns the service object for the specified reference name and ServiceReference.

Parameters:
name - The name of a reference as specified in a reference element in this component's description.
reference - The ServiceReference to a bound service. This must be a ServiceReference provided to the component via the bind or unbind method for the specified reference name.

Returns:
A service object for the referenced service or null if the specified ServiceReference is not a bound service for the specified reference name.

Throws: ComponentException - If the Service Component Runtime catches an exception while activating the bound service.

locateServices

Object[] locateServices(String name)

Returns the service objects for the specified reference name.
Parameters:
name - The name of a reference as specified in a reference element in this component's description.

Returns:
An array of service objects for the referenced service or null if the reference cardinality is 0..1 or 0..n and no bound service is available. If the reference cardinality is 0..1 or 1..1 and a bound service is available, the array will have exactly one element.

Throws:
ComponentException - If the Service Component Runtime catches an exception while activating a bound service.

---

**getBundleContext**

org.osgi.framework.BundleContext getBundleContext()

Returns the BundleContext of the bundle which contains this component.

**Returns:**
The BundleContext of the bundle containing this component.

---

**getUsingBundle**

org.osgi.framework.Bundle getUsingBundle()

If the component instance is registered as a service using the servicescope="bundle" or servicescope="prototype" attribute, then this method returns the bundle using the service provided by the component instance.

This method will return null if:

1. The component instance is not a service, then no bundle can be using it as a service.
2. The component instance is a service but did not specify the servicescope="bundle" or servicescope="prototype" attribute, then all bundles using the service provided by the component instance will share the same component instance.
3. The service provided by the component instance is not currently being used by any bundle.

**Returns:**
The bundle using the component instance as a service or null.

---

**getComponentInstance**

ComponentInstance getComponentInstance()

Returns the Component Instance object for the component instance associated with this Component Context.

**Returns:**
The Component Instance object for the component instance.

---

**enableComponent**

void enableComponent(String name)

Enables the specified component name. The specified component name must be in the same bundle as this component.
This method must return after changing the enabled state of the specified component name. Any actions that result from this, such as activating or deactivating a component configuration, must occur asynchronously to this method call.

**Parameters:**
- name - The name of a component or null to indicate all components in the bundle.

---

### disableComponent

```java
do nothing disableComponent(String name)
```

Disables the specified component name. The specified component name must be in the same bundle as this component.

This method must return after changing the enabled state of the specified component name. Any actions that result from this, such as activating or deactivating a component configuration, must occur asynchronously to this method call.

**Parameters:**
- name - The name of a component.

---

### getServiceReference

```java
getServiceReference<null> getServiceReference()
```

If the component instance is registered as a service using the `service` element, then this method returns the service reference of the service provided by this component instance.

This method will return `null` if the component instance is not registered as a service.

**Returns:**
- The `ServiceReference` object for the component instance or `null` if the component instance is not registered as a service.
Class ComponentException

org.osgi.service.component

java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            java.lang.RuntimeException
                org.osgi.service.component.ComponentException

All Implemented Interfaces:
    Serializable

public class ComponentException
extends RuntimeException

Unchecked exception which may be thrown by the Service Component Runtime.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ComponentException(String message)</td>
<td>30</td>
</tr>
<tr>
<td>Construct a new ComponentException with the specified message.</td>
<td></td>
</tr>
<tr>
<td>public ComponentException(String message, Throwable cause)</td>
<td>30</td>
</tr>
<tr>
<td>Construct a new ComponentException with the specified message and cause.</td>
<td></td>
</tr>
<tr>
<td>public ComponentException(Throwable cause)</td>
<td>31</td>
</tr>
<tr>
<td>Construct a new ComponentException with the specified cause.</td>
<td></td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throwable getCause()</td>
<td>31</td>
</tr>
<tr>
<td>Returns the cause of this exception or null if no cause was set.</td>
<td></td>
</tr>
<tr>
<td>Throwable initCause(Throwable cause)</td>
<td>31</td>
</tr>
<tr>
<td>Initializes the cause of this exception to the specified value.</td>
<td></td>
</tr>
</tbody>
</table>

Constructor Detail

ComponentException

public ComponentException(String message, Throwable cause)

Construct a new ComponentException with the specified message and cause.

Parameters:
message - The message for the exception.
cause - The cause of the exception. May be null.

ComponentException

public ComponentException(String message)

Construct a new ComponentException with the specified message.

Parameters:
message - The message for the exception.
**Class ComponentException**

*public ComponentException(Throwable cause)*

Construct a new ComponentException with the specified cause.

**Parameters:**
- cause - The cause of the exception. May be null.

### Method Detail

**getCause**

*public Throwable getCause()*

Returns the cause of this exception or null if no cause was set.

**Overrides:**
- getCause in class Throwable

**Returns:**
- The cause of this exception or null if no cause was set.

**initCause**

*public Throwable initCause(Throwable cause)*

Initializes the cause of this exception to the specified value.

**Overrides:**
- initCause in class Throwable

**Parameters:**
- cause - The cause of this exception.

**Returns:**
- This exception.

**Throws:**
- IllegalArgumentException - If the specified cause is this exception.
- IllegalStateException - If the cause of this exception has already been set.
Interface ComponentFactory

@org.osgi.annotation.versioning.ProviderType
public interface ComponentFactory

When a component is declared with the factory attribute on its component element, the Service Component Runtime will register a Component Factory service to allow new component configurations to be created and activated rather than automatically creating and activating component configuration as necessary.

ThreadSafe

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComponentInstance</td>
<td>newInstance(Dictionary&lt;String,?&gt; properties)</td>
<td>Create and activate a new component configuration.</td>
</tr>
</tbody>
</table>

Method Detail

newInstance

ComponentInstance newInstance(Dictionary<String,?> properties)

Create and activate a new component configuration. Additional properties may be provided for the component configuration.

Parameters:
- properties - Additional properties for the component configuration or null if there are no additional properties.

Returns:
- A ComponentInstance object encapsulating the component instance of the component configuration. The component configuration has been activated and, if the component specifies a service element, the component instance has been registered as a service.

Throws:
- ComponentException - If the Service Component Runtime is unable to activate the component configuration.
A ComponentInstance encapsulates a component instance of an activated component configuration. ComponentInstances are created whenever a component configuration is activated.

ComponentInstances are never reused. A new ComponentInstance object will be created when the component configuration is activated again.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>void dispose()</td>
<td>33</td>
</tr>
<tr>
<td>Object getInstance()</td>
<td>33</td>
</tr>
</tbody>
</table>

#### Method Detail

**dispose**

```java
void dispose()
```

Dispose of the component configuration for this component instance. The component configuration will be deactivated. If the component configuration has already been deactivated, this method does nothing.

**getInstance**

```java
Object getInstance()
```

Returns the component instance of the activated component configuration.

**Returns:**

The component instance or `null` if the component configuration has been deactivated.
Package org.osgi.service.component.annotations

@org.osgi.annotation.versioning.Version(value="1.3")

Service Component Annotations Package Version 1.3.

See: Description

### Enum Summary

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigurationPolicy</td>
<td>Configuration Policy for the <a href="#">Component</a> annotation.</td>
<td>42</td>
</tr>
<tr>
<td>ReferenceCardinality</td>
<td>Cardinality for the <a href="#">Reference</a> annotation.</td>
<td>53</td>
</tr>
<tr>
<td>ReferencePolicy</td>
<td>Policy for the <a href="#">Reference</a> annotation.</td>
<td>55</td>
</tr>
<tr>
<td>ReferencePolicyOption</td>
<td>Policy option for the <a href="#">Reference</a> annotation.</td>
<td>57</td>
</tr>
<tr>
<td>ReferenceScope</td>
<td>Reference scope for the <a href="#">Reference</a> annotation.</td>
<td>59</td>
</tr>
<tr>
<td>ServiceScope</td>
<td>Service scope for the <a href="#">Component</a> annotation.</td>
<td>61</td>
</tr>
</tbody>
</table>

### Annotation Types Summary

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Identify the annotated method as the activate method of a Service Component.</td>
<td>35</td>
</tr>
<tr>
<td>Component</td>
<td>Identify the annotated class as a Service Component.</td>
<td>36</td>
</tr>
<tr>
<td>Deactivate</td>
<td>Identify the annotated method as the deactivate method of a Service Component.</td>
<td>44</td>
</tr>
<tr>
<td>LookupReference</td>
<td>Define a lookup strategy reference for a <a href="#">Component</a>.</td>
<td>45</td>
</tr>
<tr>
<td>Modified</td>
<td>Identify the annotated method as the modified method of a Service Component.</td>
<td>48</td>
</tr>
<tr>
<td>Reference</td>
<td>Identify the annotated method as a bind method of a Service Component.</td>
<td>49</td>
</tr>
</tbody>
</table>

Package org.osgi.service.component.annotations Description

Service Component Annotations Package Version 1.3.

This package is not used at runtime. Annotated classes are processed by tools to generate Component Descriptions which are used at runtime.
Identify the annotated method as the `activate` method of a Service Component.

The annotated method is the activate method of the Component.

This annotation is not processed at runtime by a Service Component Runtime implementation. It must be processed by tools and used to add a Component Description to the bundle.

Since: 1.1

See Also:
“The activate attribute of the component element of a Component Description.”
# Annotation Type Component

```java
deprecated
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.TYPE)
public @interface Component
{
    String NAME;
    String[] configurationPid;
    ConfigurationPolicy configurationPolicy;
    boolean enabled;
    String factory;
    boolean immediate;
    String name;
    String[] properties;
    String[] property;
    LookupReference[] reference;
    ServiceScope scope;
    Class<?> service;
    boolean servicefactory;
    String xmlns;
}
```

Identify the annotated class as a Service Component.

The annotated class is the implementation class of the Component.

This annotation is not processed at runtime by a Service Component Runtime implementation. It must be processed by tools and used to add a Component Description to the bundle.

See Also:

- "The component element of a Component Description."

## Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>String</td>
<td>Special string representing the name of this Component.</td>
</tr>
</tbody>
</table>

## Required Element Summary

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configurationPid</td>
<td>String[]</td>
<td>The configuration PIDs for the configuration of this Component.</td>
</tr>
<tr>
<td>configurationPolicy</td>
<td>String</td>
<td>The configuration policy of this Component.</td>
</tr>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>Declares whether this Component is enabled when the bundle containing it is started.</td>
</tr>
<tr>
<td>factory</td>
<td>String</td>
<td>The factory identifier of this Component.</td>
</tr>
<tr>
<td>immediate</td>
<td>boolean</td>
<td>Declares whether this Component must be immediately activated upon becoming satisfied or whether activation should be delayed.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>The name of this Component.</td>
</tr>
<tr>
<td>properties</td>
<td>String[]</td>
<td>Property entries for this Component.</td>
</tr>
<tr>
<td>property</td>
<td>String[]</td>
<td>Properties for this Component.</td>
</tr>
<tr>
<td>reference</td>
<td>LookupReference[]</td>
<td>The lookup strategy references of this Component.</td>
</tr>
<tr>
<td>scope</td>
<td>ServiceScope</td>
<td>The service scope for the service of this Component.</td>
</tr>
<tr>
<td>service</td>
<td>Class&lt;?&gt;[]</td>
<td>The types under which to register this Component as a service.</td>
</tr>
<tr>
<td>servicefactory</td>
<td>boolean</td>
<td>Deprecated. Since 1.3.</td>
</tr>
<tr>
<td>xmlns</td>
<td>String</td>
<td>The XML name space of the Component Description for this Component.</td>
</tr>
</tbody>
</table>
Field Detail

NAME

public static final String NAME = "$"

Special string representing the name of this Component.

This string can be used in configurationPid() to specify the name of the component as a configuration PID. For example:

@Component(configurationPid={"com.acme.system", Component.NAME})

Tools creating a Component Description from this annotation must replace the special string with the actual name of this Component.

Since: 1.3

Element Detail

name

public abstract String name

The name of this Component.

If not specified, the name of this Component is the fully qualified type name of the class being annotated.

Default: 

See Also: "The name attribute of the component element of a Component Description."

service

public abstract Class<?>[] service

The types under which to register this Component as a service.

If no service should be registered, the empty value {} must be specified.

If not specified, the service types for this Component are all the directly implemented interfaces of the class being annotated.

Default: {}

See Also: "The service element of a Component Description."

factory

public abstract String factory

The factory identifier of this Component. Specifying a factory identifier makes this Component a Factory Component.

If not specified, the default is that this Component is not a Factory Component.
Default:

See Also:
"The factory attribute of the component element of a Component Description."

---

**servicefactory**

```java
public abstract boolean servicefactory
```

**Deprecated.** Declares whether this Component uses the OSGi ServiceFactory concept and each bundle using this Component's service will receive a different component instance.

This element is ignored when the `scope()` element does not have the default value. If `true`, this Component uses `bundle` service scope. If `false` or not specified, this Component uses `singleton` service scope. If the `factory()` element is specified or the `immediate()` element is specified with `true`, this element can only be specified with `false`.

Declares whether this Component uses the OSGi ServiceFactory concept and each bundle using this Component's service will receive a different component instance.

This element is ignored when the `scope()` element does not have the default value. If `true`, this Component uses `bundle` service scope. If `false` or not specified, this Component uses `singleton` service scope. If the `factory()` element is specified or the `immediate()` element is specified with `true`, this element can only be specified with `false`.

Default:
false

See Also:
"The servicefactory attribute of the service element of a Component Description."

---

**enabled**

```java
public abstract boolean enabled
```

Declares whether this Component is enabled when the bundle containing it is started.

If `true`, this Component is enabled. If `false` or not specified, this Component is disabled.

Default:
true

See Also:
"The enabled attribute of the component element of a Component Description."

---

**immediate**

```java
public abstract boolean immediate
```

Declares whether this Component must be immediately activated upon becoming satisfied or whether activation should be delayed.

If `true`, this Component must be immediately activated upon becoming satisfied. If `false`, activation of this Component is delayed. If this property is specified, its value must be `false` if the `factory()` property is also specified or must be `true` if the `service()` property is specified with an empty value.

If not specified, the default is `false` if the `factory()` property is specified or the `service()` property is not specified or specified with a non-empty value and `true` otherwise.

Default:
false
property

public abstract String[] property

Properties for this Component.

Each property string is specified as "key=value". The type of the property value can be specified in the key as key:type=value. The type must be one of the property types supported by the type attribute of the property element of a Component Description.

To specify a property with multiple values, use multiple key, value pairs. For example, "foo=bar", "foo=baz".

Default: {}

See Also: "The property element of a Component Description."

properties

public abstract String[] properties

Property entries for this Component.

Specifies the name of an entry in the bundle whose contents conform to a standard Java Properties File. The entry is read and processed to obtain the properties and their values.

Default: {}

See Also: "The properties element of a Component Description."

xmlns

public abstract String xmlns

The XML name space of the Component Description for this Component.

If not specified, the XML name space of the Component Description for this Component should be the lowest Declarative Services XML name space which supports all the specification features used by this Component.

Default: ""

See Also: "The XML name space specified for a Component Description."

configurationPolicy

public abstract ConfigurationPolicy configurationPolicy

The configuration policy of this Component.
Annotation Type Component

Controls whether component configurations must be satisfied depending on the presence of a corresponding Configuration object in the OSGi Configuration Admin service. A corresponding configuration is a Configuration object where the PID equals the name of the component.

If not specified, the OPTIONAL configuration policy is used.

Default: ConfigurationPolicy.OPTIONAL
Since: 1.1
See Also: "The configuration-policy attribute of the component element of a Component Description."

configurationPid

public abstract String[] configurationPid

The configuration PIDs for the configuration of this Component.

Each value specifies a configuration PID for this Component.

If no value is specified, the name of this Component is used as the configuration PID of this Component.

A special string ("$") can be used to specify the name of the component as a configuration PID. The NAME constant holds this special string. For example:

@Component(configurationPid={"com.acme.system", Component.NAME})

Tools creating a Component Description from this annotation must replace the special string with the actual name of this Component.

Default: {"$"}
Since: 1.2
See Also: "The configuration-pid attribute of the component element of a Component Description."

scope

public abstract ServiceScope scope

The service scope for the service of this Component.

If not specified and the deprecated servicefactory() element is not specified, the singleton service scope is used. If the factory() element is specified or the immediate() element is specified with true, this element can only be specified with the singleton service scope.

Default: ServiceScope.DEFAULT
Since: 1.3
See Also: "The scope attribute of the service element of a Component Description."

reference

public abstract LookupReference[] reference

The lookup strategy references of this Component.
To access references using the lookup strategy, `LookupReference` annotations are specified naming the reference and declaring the type of the referenced service. The referenced service can be accessed using one of the `locateService` methods of `ComponentContext`.

To access references using the event strategy, bind methods are annotated with `Reference`.

**Default:**
- `{}`

**Since:**
- 1.3

**See Also:**
- "The reference element of a Component Description."
public enum ConfigurationPolicy extends Enum<ConfigurationPolicy>

Configuration Policy for the Component annotation.

Controls whether component configurations must be satisfied depending on the presence of a corresponding Configuration object in the OSGi Configuration Admin service. A corresponding configuration is a Configuration object where the PID is the name of the component.

Since: 1.1

Enum Constant Summary

<table>
<thead>
<tr>
<th>Enum Constant</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGNORE</td>
<td>43</td>
</tr>
<tr>
<td>Always allow the component configuration to be satisfied and do not use the corresponding Configuration object even if it is present.</td>
<td></td>
</tr>
<tr>
<td>OPTIONAL</td>
<td>42</td>
</tr>
<tr>
<td>Use the corresponding Configuration object if present but allow the component to be satisfied even if the corresponding Configuration object is not present.</td>
<td></td>
</tr>
<tr>
<td>REQUIRE</td>
<td>43</td>
</tr>
<tr>
<td>There must be a corresponding Configuration object for the component configuration to become satisfied.</td>
<td></td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String toString()</td>
<td>43</td>
</tr>
<tr>
<td>static ConfigurationPolicy[]</td>
<td>43</td>
</tr>
<tr>
<td>values()</td>
<td></td>
</tr>
<tr>
<td>static ConfigurationPolicy[]</td>
<td>43</td>
</tr>
<tr>
<td>valueOf(String name)</td>
<td></td>
</tr>
</tbody>
</table>

Enum Constant Detail

**OPTIONAL**

Use the corresponding Configuration object if present but allow the component to be satisfied even if the corresponding Configuration object is not present.
REQUIRE

public static final ConfigurationPolicy REQUIRE

There must be a corresponding Configuration object for the component configuration to become satisfied.

IGNORE

public static final ConfigurationPolicy IGNORE

Always allow the component configuration to be satisfied and do not use the corresponding Configuration object even if it is present.

Method Detail

values

public static ConfigurationPolicy[] values()

valueOf

public static ConfigurationPolicy valueOf(String name)

toString

public String toString()}

Overrides: toString in class Enum
**Annotation Type Deactivate**

org.osgi.service.component.annotations

```java
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.METHOD)
public @interface Deactivate
```

Identify the annotated method as the `deactivate` method of a Service Component.

The annotated method is the `deactivate` method of the Component.

This annotation is not processed at runtime by a Service Component Runtime implementation. It must be processed by tools and used to add a Component Description to the bundle.

**Since:** 1.1

**See Also:** "The deactivate attribute of the component element of a Component Description."
**Annotation Type LookupReference**

`org.osgi.service.component.annotations`

```java
@Retention(value=RetentionPolicy.CLASS)
@Target(value={})
public @interface LookupReference
```

Define a lookup strategy reference for a `Component`

The referenced service can be accessed using one of the `locateService` methods of `ComponentContext`.

This annotation is not processed at runtime by a Service Component Runtime implementation. It must be processed by tools and used to add a Component Description to the bundle.

In the generated Component Description for a component, the references must be ordered in ascending lexicographical order (using `String.compareTo`) of the reference `names`.

**Since:** 1.3

**See Also:** "The reference element of a Component Description."

### Required Element Summary

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cardinality</td>
<td>The cardinality of the reference.</td>
<td>46</td>
</tr>
<tr>
<td>name</td>
<td>The name of this reference.</td>
<td>45</td>
</tr>
<tr>
<td>policy</td>
<td>The policy for the reference.</td>
<td>46</td>
</tr>
<tr>
<td>policyOption</td>
<td>The policy option for the reference.</td>
<td>46</td>
</tr>
<tr>
<td>scope</td>
<td>The requested service scope for this Reference.</td>
<td>47</td>
</tr>
<tr>
<td>service</td>
<td>The type of the service to bind to this reference.</td>
<td>45</td>
</tr>
<tr>
<td>target</td>
<td>The target filter for the reference.</td>
<td>46</td>
</tr>
</tbody>
</table>

### Element Detail

**name**

```java
public abstract String name
```

The name of this reference.

**See Also:** "The name attribute of the reference element of a Component Description."

**service**

```java
public abstract Class<?> service
```

The type of the service to bind to this reference.
cardinality

```java
class ReferenceCardinality {
    public abstract String cardinality;
    // The cardinality of the reference.
    // If not specified, the reference has a 1..1 cardinality.
    // Default: ReferenceCardinality.MANDATORY
    // See Also: "The cardinality attribute of the reference element of a Component Description."
}
```

policy

```java
class ReferencePolicy {
    public abstract String policy;
    // The policy for the reference.
    // If not specified, the STATIC reference policy is used.
    // Default: ReferencePolicy.STATIC
    // See Also: "The policy attribute of the reference element of a Component Description."
}
```

target

```java
class ReferencePolicyOption {
    public abstract String policyOption;
    // The target filter for the reference.
    // Default: ""
    // See Also: "The target attribute of the reference element of a Component Description."
}
```
scope

public abstract ReferenceScope scope

The requested service scope for this Reference.

If not specified, the bundle service scope is requested.

Default: ReferenceScope.BUNDLE

See Also: “The scope attribute of the reference element of a Component Description.”
### Annotation Type Modified

```java
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.METHOD)
public @interface Modified
```

Identify the annotated method as the modified method of a Service Component.

The annotated method is the modified method of the Component.

This annotation is not processed at runtime by a Service Component Runtime implementation. It must be processed by tools and used to add a Component Description to the bundle.

**Since:** 1.1

**See Also:** "The modified attribute of the component element of a Component Description."
Identify the annotated method as a bind method of a Service Component.

The annotated method is a bind method of the Component.

This annotation is not processed at runtime by a Service Component Runtime implementation. It must be processed by tools and used to add a Component Description to the bundle.

In the generated Component Description for a component, the references must be ordered in ascending lexicographical order (using String.compareTo) of the reference names.

See Also:
"The reference element of a Component Description."

### Required Element Summary

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cardinality</td>
<td>The cardinality of the reference.</td>
<td>50</td>
</tr>
<tr>
<td>name</td>
<td>The name of this reference.</td>
<td>49</td>
</tr>
<tr>
<td>policy</td>
<td>The policy for the reference.</td>
<td>50</td>
</tr>
<tr>
<td>policyOption</td>
<td>The policy option for the reference.</td>
<td>51</td>
</tr>
<tr>
<td>scope</td>
<td>The requested service scope for this Reference.</td>
<td>51</td>
</tr>
<tr>
<td>service</td>
<td>The type of the service to bind to this reference.</td>
<td>50</td>
</tr>
<tr>
<td>target</td>
<td>The target filter for the reference.</td>
<td>50</td>
</tr>
<tr>
<td>unbind</td>
<td>The name of the unbind method which is associated with the annotated bind method.</td>
<td>50</td>
</tr>
<tr>
<td>updated</td>
<td>The name of the updated method which is associated with the annotated bind method.</td>
<td>51</td>
</tr>
</tbody>
</table>

### Element Detail

**name**

```java
public abstract String name
```

The name of this reference.

If not specified, the name of this reference is based upon the name of the method being annotated. If the method name begins with bind, set or add, that is removed.

**Default:**

**See Also:**
"The name attribute of the reference element of a Component Description."
service

```java
public abstract Class<? extends Object> service
```

The type of the service to bind to this reference.

If not specified, the type of the service to bind is based upon the type of the first argument of the method being annotated.

**Default:**
- `Object.class`

**See Also:**
- “The interface attribute of the reference element of a Component Description.”

---

cardinality

```java
public abstract ReferenceCardinality cardinality
```

The cardinality of the reference.

If not specified, the reference has a `1..1` cardinality.

**Default:**
- `ReferenceCardinality.MANDATORY`

**See Also:**
- “The cardinality attribute of the reference element of a Component Description.”

---

policy

```java
public abstract ReferencePolicy policy
```

The policy for the reference.

If not specified, the `STATIC` reference policy is used.

**Default:**
- `ReferencePolicy.STATIC`

**See Also:**
- “The policy attribute of the reference element of a Component Description.”

---

target

```java
public abstract String target
```

The target filter for the reference.

**Default:**
- `""`

**See Also:**
- “The target attribute of the reference element of a Component Description.”

---

unbind

```java
public abstract String unbind
```
The name of the unbind method which is associated with the annotated bind method.

To declare no unbind method, the value "-" must be used.

If not specified, the name of the unbind method is derived from the name of the annotated bind method. If the annotated method name begins with bind, set or add, that is replaced with unbind, unset or remove, respectively, to derive the unbind method name. Otherwise, un is prefixed to the annotated method name to derive the unbind method name. The unbind method is only set if the component type contains a method with the derived name.

Default: ""

See Also:
"The unbind attribute of the reference element of a Component Description."

**policyOption**

```java
public abstract ReferencePolicyOption policyOption
```

The policy option for the reference.

If not specified, the RELUCTANT reference policy option is used.

Default: ReferencePolicyOption.RELUCTANT

Since: 1.2

See Also:
"The policy-option attribute of the reference element of a Component Description."

**updated**

```java
public abstract String updated
```

The name of the updated method which is associated with the annotated bind method.

To declare no updated method, the value "-" must be used.

If not specified, the name of the updated method is derived from the name of the annotated bind method. If the annotated method name begins with bind, set or add, that is replaced with updated to derive the updated method name. Otherwise, updated is prefixed to the annotated method name to derive the updated method name. The updated method is only set if the component type contains a method with the derived name.

Default: ""

Since: 1.2

See Also:
"The updated attribute of the reference element of a Component Description."

**scope**

```java
public abstract ReferenceScope scope
```

The requested service scope for this Reference.

If not specified, the bundle service scope is requested.
Default: ReferenceScope.BUNDLE
Since: 1.3
See Also:
"The scope attribute of the reference element of a Component Description."
Enum ReferenceCardinality

org.osgi.service.component.annotations

describes the reference policy. It needs to only be used on the @Reference annotation.

java.lang.Object

java.lang.Enum<ReferenceCardinality>

org.osgi.service.component.annotations.ReferenceCardinality

All Implemented Interfaces:
Comparable<ReferenceCardinality>, Serializable

public enum ReferenceCardinality
extends Enum<ReferenceCardinality>

Cardinality for the Reference annotation.

Specifies if the reference is optional and if the component implementation support a single bound service or multiple bound services.

Enum Constant Summary

<table>
<thead>
<tr>
<th>AT_LEAST_ONE</th>
<th>The reference is mandatory and multiple.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MANDATORY</td>
<td>The reference is mandatory and unary.</td>
</tr>
<tr>
<td>MULTIPLE</td>
<td>The reference is optional and multiple.</td>
</tr>
<tr>
<td>OPTIONAL</td>
<td>The reference is optional and unary.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>String toString()</th>
<th></th>
</tr>
</thead>
</table>

static ReferenceCardinality valueOf(String name)

static values()

Enum Constant Detail

OPTIONAL

public static final ReferenceCardinality OPTIONAL

The reference is optional and unary. That is, the reference has a cardinality of 0..1.

MANDATORY

public static final ReferenceCardinality MANDATORY

The reference is mandatory and unary. That is, the reference has a cardinality of 1..1.
**MULTIPLE**

public static final ReferenceCardinality MULTIPLE

The reference is optional and multiple. That is, the reference has a cardinality of 0..n.

**AT_LEAST_ONE**

public static final ReferenceCardinality AT_LEAST_ONE

The reference is mandatory and multiple. That is, the reference has a cardinality of 1..n.

### Method Detail

**values**

public static ReferenceCardinality[] values()

**valueOf**

public static ReferenceCardinality valueOf(String name)

**toString**

public String toString()

**Overrides:**

toString in class Enum
Enum ReferencePolicy

org.osgi.service.component.annotations

java.lang.Object
  java.lang.Enum<ReferencePolicy>
    org.osgi.service.component.annotations.ReferencePolicy

AllImplementedInterfaces:
  Comparable<ReferencePolicy>, Serializable

public enum ReferencePolicy
  extends Enum<ReferencePolicy>

PolicyfortheReferenceannotation.

EnumConstantSummary

<table>
<thead>
<tr>
<th>Const</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DYNAMIC</td>
<td>The dynamic policy is slightly more complex since the component implementation must properly handle changes in the set of bound services.</td>
</tr>
<tr>
<td>STATIC</td>
<td>The static policy is the most simple policy and is the default policy.</td>
</tr>
</tbody>
</table>

MethodSummary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String toString()</td>
<td>56</td>
</tr>
<tr>
<td>static ReferencePolicy.valueOf(String name)</td>
<td>56</td>
</tr>
<tr>
<td>static ReferencePolicy[] values()</td>
<td>56</td>
</tr>
</tbody>
</table>

EnumConstantDetail

STATIC

public static final ReferencePolicy STATIC

The static policy is the most simple policy and is the default policy. A component instance never sees any of the dynamics. Component configurations are deactivated before any bound service for a reference having a static policy becomes unavailable. If a target service is available to replace the bound service which became unavailable, the component configuration must be reactivated and bound to the replacement service.

DYNAMIC

public static final ReferencePolicy DYNAMIC

The dynamic policy is slightly more complex since the component implementation must properly handle changes in the set of bound services. With the dynamic policy, SCR can change the set of bound services without deactivating a component configuration. If the component uses the event strategy to access services, then the component instance will be notified of changes in the set of bound services by calls to the bind and unbind methods.
Method Detail

values

public static ReferencePolicy[] values()

valueOf

public static ReferencePolicy valueOf(String name)

toString

public String toString()

Overrides:
toString in class Enum
Enum ReferencePolicyOption

org.osgi.service.component.annotations

java.lang.Object
   java.lang.Enum<ReferencePolicyOption>
   org.osgi.service.component.annotations.ReferencePolicyOption

All Implemented Interfaces:
   Comparable<ReferencePolicyOption>, Serializable

public enum ReferencePolicyOption
   extends Enum<ReferencePolicyOption>

Policy option for the Reference annotation.

Since: 1.2

Enum Constant Summary

<table>
<thead>
<tr>
<th>Constant</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREEDY</td>
<td>57</td>
</tr>
<tr>
<td>RELUCTANT</td>
<td>57</td>
</tr>
</tbody>
</table>

- **GREEDY**
  The greedy policy option is a valid policy option for both static and dynamic reference policies.

- **RELUCTANT**
  The reluctant policy option is the default policy option for both static and dynamic reference policies.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String toString()</td>
<td>58</td>
</tr>
<tr>
<td>static ReferencePolicyOption[]</td>
<td>58</td>
</tr>
<tr>
<td>static ReferencePolicyOption[]</td>
<td>58</td>
</tr>
<tr>
<td>static values()</td>
<td>58</td>
</tr>
</tbody>
</table>

Enum Constant Detail

**RELUCTANT**

public static final ReferencePolicyOption RELUCTANT

The reluctant policy option is the default policy option for both static and dynamic reference policies. When a new target service for a reference becomes available, references having the reluctant policy option for the static policy or the dynamic policy with a unary cardinality will ignore the new target service. References having the dynamic policy with a multiple cardinality will bind the new target service.

**GREEDY**

public static final ReferencePolicyOption GREEDY

The greedy policy option is a valid policy option for both static and dynamic reference policies. When a new target service for a reference becomes available, references having the greedy policy option will bind the new target service.
Enum ReferenceScope

Method Detail

values

public static ReferencePolicyOption[] values()

valueOf

public static ReferencePolicyOption.valueOf(String name)

toString

public String toString()

Overrides:
toString in class Enum
**Enum ReferenceScope**

`org.osgi.service.component.annotations`

java.lang.Object  
  | java.lang.Enum<ReferenceScope>  
  | org.osgi.service.component.annotations.ReferenceScope

All Implemented Interfaces:  
  Comparable<ReferenceScope>, Serializable

public enum ReferenceScope  
  extends Enum<ReferenceScope>

Reference scope for the `Reference` annotation.

Since:  
1.3

---

### Enum Constant Summary

<table>
<thead>
<tr>
<th>ReferenceScope</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUNDLE</td>
<td>59</td>
</tr>
<tr>
<td>PROTOTYPE</td>
<td>59</td>
</tr>
</tbody>
</table>

**BUNDLE**  
A single service object is used for all references to the service in this bundle.

**PROTOTYPE**  
If the referenced service has prototype service scope, then each instance of the component with this reference can receive a unique instance of the service.

---

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toString()</code></td>
<td>60</td>
</tr>
<tr>
<td><code>valueOf(String name)</code></td>
<td>60</td>
</tr>
<tr>
<td><code>values()</code></td>
<td>60</td>
</tr>
</tbody>
</table>

---

### Enum Constant Detail

**BUNDLE**

public static final `ReferenceScope` BUNDLE  
A single service object is used for all references to the service in this bundle.

**PROTOTYPE**

public static final `ReferenceScope` PROTOTYPE  
If the referenced service has prototype service scope, then each instance of the component with this reference can receive a unique instance of the service. If the referenced service does not have prototype service scope, then no service object will be received.
values

public static ReferenceScope[] values()

valueOf

public static ReferenceScope valueOf(String name)

toString

public String toString()

Overrides:

toString in class Enum
Enum ServiceScope

org.osgi.service.component.annotations

java.lang.Object

java.lang.Enum<ServiceScope>

org.osgi.service.component.annotations.ServiceScope

All Implemented Interfaces:
Comparable<ServiceScope>, Serializable

public enum ServiceScope
extends Enum<ServiceScope>

Service scope for the Component annotation.

Since: 1.3

Enum Constant Summary

<table>
<thead>
<tr>
<th>ServiceScope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUNDLE</td>
<td>When the component is registered as a service, it will be registered as a bundle scope service and an instance of the component will be created for each bundle using the service.</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Default element value for annotation.</td>
</tr>
<tr>
<td>PROTOTYPE</td>
<td>When the component is registered as a service, it will be registered as a prototype scope service.</td>
</tr>
<tr>
<td>SINGLETON</td>
<td>When the component is registered as a service, it will be registered as a bundle scope service but only a single instance of the component will be used for all bundles using the service.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String toString()</td>
<td></td>
</tr>
<tr>
<td>static ServiceScope values()</td>
<td></td>
</tr>
</tbody>
</table>

Enum Constant Detail

SINGLETON

public static final ServiceScope SINGLETON

When the component is registered as a service, it will be registered as a bundle scope service but only a single instance of the component will be used for all bundles using the service.

BUNDLE

public static final ServiceScope BUNDLE
When the component is registered as a service, it will be registered as a bundle scope service and an instance of the component will be created for each bundle using the service.

**PROTOTYPE**

```java
public static final ServiceScope PROTOTYPE
```

When the component is registered as a service, it will be registered as a prototype scope service.

**DEFAULT**

```java
public static final ServiceScope DEFAULT
```

Default element value for annotation. This is used to distinguish the default value for an element and should not otherwise be used.

## Method Detail

### values

```java
public static ServiceScope[] values()
```

### valueOf

```java
public static ServiceScope valueOf(String name)
```

### toString

```java
public String toString()
```

*Overrides:* `toString in class Enum`
Package org.osgi.service.component.runtime

@org.osgi.annotation.versioning.Version(value="1.3")

Service Component Runtime Package Version 1.3.

See: 

Description

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ServiceComponentRuntime</code></td>
<td>The <code>ServiceComponentRuntime</code> service represents the Declarative Services main controller also known as the Service Component Runtime or SCR for short.</td>
</tr>
</tbody>
</table>

Package org.osgi.service.component.runtime Description

Service Component Runtime Package Version 1.3.

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.component.runtime; version="[1.3,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.service.component.runtime; version="[1.3,1.4)"
Interface ServiceComponentRuntime

@org.osgi.annotation.versioning.ProviderType
class ServiceComponentRuntime

public interface ServiceComponentRuntime

The ServiceComponentRuntime service represents the Declarative Services main controller also known as the Service Component Runtime or SCR for short. It provides access to the components managed by the Service Component Runtime.

This service differentiates between a ComponentDescriptionDTO and a ComponentConfigurationDTO. A ComponentDescriptionDTO is a representation of a declared component description. A ComponentConfigurationDTO is a representation of an actual instance of a declared component description parameterized by component properties.

Access to this service requires the ServicePermission[ServiceComponentRuntime, GET] permission. It is intended that only administrative bundles should be granted this permission to limit access to the potentially intrusive methods provided by this service.

Since: 1.3
ThreadSafe

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.util.promise.Promise&lt;Void&gt; disableComponent(ComponentDescriptionDTO description)</td>
<td>66</td>
</tr>
<tr>
<td>Enables the specified component description.</td>
<td></td>
</tr>
<tr>
<td>org.osgi.util.promise.Promise&lt;Void&gt; enableComponent(ComponentDescriptionDTO description)</td>
<td>66</td>
</tr>
<tr>
<td>Enables the specified component description.</td>
<td></td>
</tr>
<tr>
<td>Collection&lt;ComponentConfigurationDTO&gt; getComponentConfigurationDTOs(ComponentDescriptionDTO description)</td>
<td>65</td>
</tr>
<tr>
<td>Returns the component configurations for the specified component description.</td>
<td></td>
</tr>
<tr>
<td>ComponentDescriptionDTO getComponentDescriptionDTO(org.osgi.framework.Bundle bundle, String name)</td>
<td>65</td>
</tr>
<tr>
<td>Returns the ComponentDescriptionDTO declared with the specified name by the specified bundle.</td>
<td></td>
</tr>
<tr>
<td>Collection&lt;ComponentDescriptionDTO&gt; getComponentDescriptionDTOs(org.osgi.framework.Bundle... bundles)</td>
<td>64</td>
</tr>
<tr>
<td>Returns the component descriptions declared by the specified active bundles.</td>
<td></td>
</tr>
<tr>
<td>boolean isEnabled(ComponentDescriptionDTO description)</td>
<td>65</td>
</tr>
<tr>
<td>Returns whether the specified component description is currently enabled.</td>
<td></td>
</tr>
</tbody>
</table>

Method Detail
gGetComponentDescriptionDTOs

Collection<ComponentDescriptionDTO> getComponentDescriptionDTOs(org.osgi.framework.Bundle... bundles)

Returns the component descriptions declared by the specified active bundles.

Only component descriptions from active bundles are returned. If the specified bundles have no declared components or are not active, an empty collection is returned.
Class ComponentConfigurationDTO

**Parameters:**
- bundles - The bundles whose declared component descriptions are to be returned. Specifying no bundles, or the equivalent of an empty Bundle array, will return the declared component descriptions from all active bundles.

**Returns:**
The declared component descriptions of the specified active bundles. An empty collection is returned if there are no component descriptions for the specified active bundles.

---

**GetComponentDescriptionDTO**

ComponentDescriptionDTO getComponentDescriptionDTO(org.osgi.framework.Bundle bundle, String name)

Returns the ComponentDescriptionDTO declared with the specified name by the specified bundle. Only component descriptions from active bundles are returned. null if no such component is declared by the given bundle or the bundle is not active.

**Parameters:**
- bundle - The bundle declaring the component description. Must not be null.
- name - The name of the component description. Must not be null.

**Returns:**
The declared component description or null if the specified bundle is not active or does not declare a component description with the specified name.

---

**GetComponentConfigurationDTOs**

Collection<ComponentConfigurationDTO> getComponentConfigurationDTOs(ComponentDescriptionDTO description)

Returns the component configurations for the specified component description.

**Parameters:**
- description - The component description. Must not be null.

**Returns:**
A collection containing a snapshot of the current component configurations for the specified component description. An empty collection is returned if there are none.

---

**isComponentEnabled**

boolean isComponentEnabled(ComponentDescriptionDTO description)

Returns whether the specified component description is currently enabled.

The enabled state of a component description is initially set by the enabled attribute of the component description.

**Parameters:**
- description - The component description. Must not be null.

**Returns:**
true if the specified component description is currently enabled. Otherwise, false.

**See Also:**
- enableComponent(ComponentDescriptionDTO),
- disableComponent(ComponentDescriptionDTO),
- ComponentContext.disableComponent(String),
- ComponentContext.enableComponent(String)
enableComponent

org.osgi.util.promise.Promise<Void> enableComponent(ComponentDescriptionDTO description)

Enables the specified component description.

If the specified component description is currently enabled, this method has no effect.

This method must return after changing the enabled state of the specified component description. Any actions that result from this, such as activating or deactivating a component configuration, must occur asynchronously to this method call.

Parameters:
description - The component description to enable. Must not be null.

Returns:
A promise that will be resolved when the actions that result from changing the enabled state of the specified component have completed.

See Also:
isComponentEnabled(ComponentDescriptionDTO)

disableComponent

org.osgi.util.promise.Promise<Void> disableComponent(ComponentDescriptionDTO description)

Disables the specified component description.

If the specified component description is currently disabled, this method has no effect.

This method must return after changing the enabled state of the specified component description. Any actions that result from this, such as activating or deactivating a component configuration, must occur asynchronously to this method call.

Parameters:
description - The component description to disable. Must not be null.

Returns:
A promise that will be resolved when the actions that result from changing the enabled state of the specified component have completed.

See Also:
isComponentEnabled(ComponentDescriptionDTO)
Package org.osgi.service.component.runtime.dto

@org.osgi.annotation.versioning.Version(value="1.3")

Service Component Runtime Data Transfer Objects Package Version 1.3.

See: Description

Class Summary

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComponentConfigurationDTO</td>
<td>68</td>
</tr>
<tr>
<td>A representation of an actual instance of a declared component description parameterized by component properties.</td>
<td></td>
</tr>
<tr>
<td>ComponentDescriptionDTO</td>
<td>71</td>
</tr>
<tr>
<td>A representation of a declared component description.</td>
<td></td>
</tr>
<tr>
<td>ReferenceDTO</td>
<td>75</td>
</tr>
<tr>
<td>A representation of a declared reference to a service.</td>
<td></td>
</tr>
<tr>
<td>SatisfiedReferenceDTO</td>
<td>78</td>
</tr>
<tr>
<td>A representation of a satisfied reference.</td>
<td></td>
</tr>
<tr>
<td>UnsatisfiedReferenceDTO</td>
<td>80</td>
</tr>
<tr>
<td>A representation of an unsatisfied reference.</td>
<td></td>
</tr>
</tbody>
</table>

Service Component Runtime Data Transfer Objects Package Version 1.3.

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.component.runtime.dto; version="[1.3,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.service.component.runtime.dto; version="[1.3,1.4)"
Class ComponentConfigurationDTO

org.osgi.service.component.runtime.dto

java.lang.Object
    org.osgi.dto.DTO
        org.osgi.service.component.runtime.dto.ComponentConfigurationDTO

public class ComponentConfigurationDTO
extends org.osgi.dto.DTO

A representation of an actual instance of a declared component description parameterized by component properties.

Since: 1.3
NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int ACTIVE</td>
<td>69</td>
</tr>
<tr>
<td>long id</td>
<td>69</td>
</tr>
<tr>
<td>Map&lt;String,Object&gt; properties</td>
<td>69</td>
</tr>
<tr>
<td>static int SATISFIED</td>
<td>69</td>
</tr>
<tr>
<td>int state</td>
<td>69</td>
</tr>
<tr>
<td>static int UNSATISFIED_CONFIGURATION</td>
<td>68</td>
</tr>
<tr>
<td>static int UNSATISFIED_REFERENCE</td>
<td>69</td>
</tr>
</tbody>
</table>

Field Detail

UNSATISFIED_CONFIGURATION

public static final int UNSATISFIED_CONFIGURATION = 1
The component configuration is unsatisfied due to a missing required configuration.

**UNSATISFIED_REFERENCE**

```java
public static final int UNSATISFIED_REFERENCE = 2
```

The component configuration is unsatisfied due to an unsatisfied reference.

**SATISFIED**

```java
public static final int SATISFIED = 4
```

The component configuration is satisfied.

Any `services` declared by the component description are registered.

**ACTIVE**

```java
public static final int ACTIVE = 8
```

The component configuration is active.

This is the normal operational state of a component configuration.

**description**

```java
public ComponentDescriptionDTO description
```

The representation of the component configuration's component description.

**id**

```java
public long id
```

The id of the component configuration.

The id is a non-persistent, unique value assigned at runtime. The id is also available as the `component.id` component property.

**state**

```java
public int state
```

The current state of the component configuration.

This is one of `UNSATISFIED_CONFIGURATION`, `UNSATISFIED_REFERENCE`, `SATISFIED` or `ACTIVE`.

**properties**

```java
public Map<String, Object> properties
```
Class ReferenceDTO

The component properties for the component configuration.

See Also:
ComponentContext.getProperties()

satisfiedReferences

public SatisfiedReferenceDTO[] satisfiedReferences

The satisfied references.

Each SatisfiedReferenceDTO in the array represents a satisfied reference of the component configuration. The array must be empty if the component configuration has no satisfied references.

unsatisfiedReferences

public UnsatisfiedReferenceDTO[] unsatisfiedReferences

The unsatisfied references.

Each UnsatisfiedReferenceDTO in the array represents an unsatisfied reference of the component configuration. The array must be empty if the component configuration has no unsatisfied references.

Constructor Detail

ComponentConfigurationDTO

public ComponentConfigurationDTO()
Class ComponentDescriptionDTO

org.osgi.service.component.runtime.dto

java.lang.Object
   | org.osgi.dto.DTO
     | org.osgi.service.component.runtime.dto.ComponentDescriptionDTO

public class ComponentDescriptionDTO
extends org.osgi.dto.DTO

A representation of a declared component description.

Since: 1.3
NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String activate</td>
<td>73</td>
</tr>
<tr>
<td>The name of the activate method.</td>
<td></td>
</tr>
<tr>
<td>org.osgi.framework.dto.BundleDTO bundle</td>
<td>72</td>
</tr>
<tr>
<td>The bundle declaring the component description.</td>
<td></td>
</tr>
<tr>
<td>String[]{] configurationPid configurationPolicy</td>
<td>74</td>
</tr>
<tr>
<td>The configuration pids. The configuration policy.</td>
<td></td>
</tr>
<tr>
<td>String deactivate</td>
<td>73</td>
</tr>
<tr>
<td>The name of the deactivate method.</td>
<td></td>
</tr>
<tr>
<td>boolean defaultEnabled</td>
<td>72</td>
</tr>
<tr>
<td>The initial enabled state.</td>
<td></td>
</tr>
<tr>
<td>String factory</td>
<td>72</td>
</tr>
<tr>
<td>The component factory name.</td>
<td></td>
</tr>
<tr>
<td>boolean immediate</td>
<td>73</td>
</tr>
<tr>
<td>The immediate state.</td>
<td></td>
</tr>
<tr>
<td>String implementationClass</td>
<td>72</td>
</tr>
<tr>
<td>The fully qualified name of the implementation class.</td>
<td></td>
</tr>
<tr>
<td>String modified</td>
<td>74</td>
</tr>
<tr>
<td>The name of the modified method.</td>
<td></td>
</tr>
<tr>
<td>String name</td>
<td>72</td>
</tr>
<tr>
<td>The name of the component.</td>
<td></td>
</tr>
<tr>
<td>Map&lt;String,Object&gt; properties</td>
<td>73</td>
</tr>
<tr>
<td>The declared component properties.</td>
<td></td>
</tr>
<tr>
<td>ReferencesDTO[] references</td>
<td>73</td>
</tr>
<tr>
<td>The referenced services.</td>
<td></td>
</tr>
<tr>
<td>String scope</td>
<td>72</td>
</tr>
<tr>
<td>The service scope.</td>
<td></td>
</tr>
<tr>
<td>String[]{] serviceInterfaces</td>
<td>73</td>
</tr>
<tr>
<td>The fully qualified names of the service interfaces.</td>
<td></td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComponentDescriptionDTO()</td>
<td>74</td>
</tr>
</tbody>
</table>
**Field Detail**

**name**

```java
public String name
```

The name of the component.

This is declared in the `name` attribute of the `component` element. This must be the default name if the component description does not declare a name.

**bundle**

```java
public org.osgi.framework.dto.BundleDTO bundle
```

The bundle declaring the component description.

**factory**

```java
public String factory
```

The component factory name.

This is declared in the `factory` attribute of the `component` element. This must be `null` if the component description is not declared as a component factory.

**scope**

```java
public String scope
```

The service scope.

This is declared in the `scope` attribute of the `service` element. This must be `null` if the component description does not declare any service interfaces.

**implementationClass**

```java
public String implementationClass
```

The fully qualified name of the implementation class.

This is declared in the `class` attribute of the `implementation` element.

**defaultEnabled**

```java
public boolean defaultEnabled
```

The initial enabled state.
This is declared in the enabled attribute of the component element.

immediate

public boolean immediate

The immediate state.

This is declared in the immediate attribute of the component element.

serviceInterfaces

public String[] serviceInterfaces

The fully qualified names of the service interfaces.

These are declared in the interface attribute of the provide elements. The array must be empty if the component description does not declare any service interfaces.

properties

public Map<String, Object> properties

The declared component properties.

These are declared in the property and properties elements.

references

public ReferenceDTO[] references

The referenced services.

These are declared in the reference elements. The array must be empty if the component description does not declare references to any services.

activate

public String activate

The name of the activate method.

This is declared in the activate attribute of the component element. This must be null if the component description does not declare an activate method name.

defactivate

public String deactivate

The name of the deactivate method.
This is declared in the `deactivate` attribute of the `component` element. This must be `null` if the component description does not declare a deactivate method name.

---

### modified

```
public String modified
```

The name of the modified method.

This is declared in the `modified` attribute of the `component` element. This must be `null` if the component description does not declare a modified method name.

---

### configurationPolicy

```
public String configurationPolicy
```

The configuration policy.

This is declared in the `configuration-policy` attribute of the `component` element. This must be the default configuration policy if the component description does not declare a configuration policy.

---

### configurationPid

```
public String[] configurationPid
```

The configuration pids.

These are declared in the `configuration-pid` attribute of the `component` element. This must contain the default configuration pid if the component description does not declare a configuration pid.

---

## Constructor Detail

### ComponentDescriptionDTO

```
public ComponentDescriptionDTO()
```
public class ReferenceDTO
extends org.osgi.dto.DTO

A representation of a declared reference to a service.

Since: 1.3
NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>bind</td>
<td>String</td>
<td>The name of the bind method of the reference.</td>
<td>76</td>
</tr>
<tr>
<td>cardinality</td>
<td>String</td>
<td>The cardinality of the reference.</td>
<td>76</td>
</tr>
<tr>
<td>interfaceName</td>
<td>String</td>
<td>The service interface of the reference.</td>
<td>76</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>The name of the reference.</td>
<td>75</td>
</tr>
<tr>
<td>policy</td>
<td>String</td>
<td>The policy of the reference.</td>
<td>76</td>
</tr>
<tr>
<td>policyOption</td>
<td>String</td>
<td>The policy option of the reference.</td>
<td>76</td>
</tr>
<tr>
<td>scope</td>
<td>String</td>
<td>The scope of the reference.</td>
<td>77</td>
</tr>
<tr>
<td>target</td>
<td>String</td>
<td>The target of the reference.</td>
<td>76</td>
</tr>
<tr>
<td>unbind</td>
<td>String</td>
<td>The name of the unbind method of the reference.</td>
<td>77</td>
</tr>
<tr>
<td>updated</td>
<td>String</td>
<td>The name of the updated method of the reference.</td>
<td>77</td>
</tr>
</tbody>
</table>

Constructor Summary

ReferenceDTO()

Methods inherited from class org.osgi.dto.DTO

toString

Field Detail

name

public String name

The name of the reference.
This is declared in the name attribute of the reference element. This must be the default name if the component description does not declare a name for the reference.

interfaceName

public String interfaceName

The service interface of the reference.

This is declared in the interface attribute of the reference element.

cardinality

public String cardinality

The cardinality of the reference.

This is declared in the cardinality attribute of the reference element. This must be the default cardinality if the component description does not declare a cardinality for the reference.

policy

public String policy

The policy of the reference.

This is declared in the policy attribute of the reference element. This must be the default policy if the component description does not declare a policy for the reference.

policyOption

public String policyOption

The policy option of the reference.

This is declared in the policy-option attribute of the reference element. This must be the default policy option if the component description does not declare a policy option for the reference.

target

public String target

The target of the reference.

This is declared in the target attribute of the reference element. This must be null if the component description does not declare a target for the reference.

bind

public String bind

The name of the bind method of the reference.
This is declared in the `bind` attribute of the `reference` element. This must be `null` if the component description does not declare a bind method for the reference.

###unbind

```java
public String bind
```

The name of the unbind method of the reference.

This is declared in the `unbind` attribute of the `reference` element. This must be `null` if the component description does not declare an unbind method for the reference.

###updated

```java
public String updated
```

The name of the updated method of the reference.

This is declared in the `updated` attribute of the `reference` element. This must be `null` if the component description does not declare an updated method for the reference.

###scope

```java
public String scope
```

The scope of the reference.

This is declared in the `scope` attribute of the `reference` element. This must be the default scope if the component description does not declare a scope for the reference.

####Constructor Detail

**ReferenceDTO**

```java
public ReferenceDTO()
```
public class SatisfiedReferenceDTO
extends org.osgi.dto.DTO

A representation of a satisfied reference.

Since: 1.3
NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.framework.dto.ServiceReferenceDTO[] boundServices</td>
<td>79</td>
</tr>
<tr>
<td>String name</td>
<td>78</td>
</tr>
<tr>
<td>String target</td>
<td>78</td>
</tr>
</tbody>
</table>

The bound services.
The name of the declared reference.
The target property of the satisfied reference.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SatisfiedReferenceDTO()</td>
<td>79</td>
</tr>
</tbody>
</table>

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

name

public String name

The name of the declared reference.
This is declared in the name attribute of the reference element of the component description.

See Also:
ReferenceDTO.name

target

public String target

The target property of the satisfied reference.
This is the value of the component property whose name is the concatenation of the declared reference name and ".target". This must be null if no target property is set for the reference.

### boundServices

**public org.osgi.framework.dto.ServiceReferenceDTO[] boundServices**

The bound services.

Each `org.osgi.framework.dto.ServiceReferenceDTO` in the array represents a service bound to the satisfied reference. The array must be empty if there are no bound services.

---

**Constructor Detail**

### SatisfiedReferenceDTO

**public SatisfiedReferenceDTO()**
public class UnsatisfiedReferenceDTO
extends org.osgi.dto.DTO

A representation of an unsatisfied reference.

Since: 1.3
NotThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>The name of the declared reference.</td>
<td>80</td>
</tr>
<tr>
<td>target</td>
<td>String</td>
<td>The target property of the unsatisfied reference.</td>
<td>80</td>
</tr>
<tr>
<td>targetServices</td>
<td>org.osgi.framework.dto.ServiceReferenceDTO[]</td>
<td>The target services.</td>
<td>81</td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnsatisfiedReferenceDTO()</td>
<td></td>
<td>81</td>
</tr>
</tbody>
</table>

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

name

public String name

The name of the declared reference.

This is declared in the name attribute of the reference element of the component description.

See Also: ReferenceDTO.name

target

public String target

The target property of the unsatisfied reference.
Class UnsatisfiedReferenceDTO

This is the value of the component property whose name is the concatenation of the declared reference name and ".target". This must be null if no target property is set for the reference.

targetServices

public org.osgi.framework.dto.ServiceReferenceDTO[] targetServices

The target services.

Each org.osgi.framework.dto.ServiceReferenceDTO in the array represents a target service for the reference. The array must be empty if there are no target services. The upper bound on the number of target services in the array is the upper bound on the cardinality of the reference.

Constructor Detail

UnsatisfiedReferenceDTO

public UnsatisfiedReferenceDTO()
Yet full support is problematic because annotations are evaluated at build time based on build time dependencies available. Later at run time the static declarations are used to define properties, bind references and expose services. If a base class is modified between some expectations of that base class may not be met.

Consider for example a component `Extender` extending from the `Base` component. The `Base` component has an optional reference to `Service S1` at build time. At deployment time a new version of the `Base` component is deployed which besides the optional reference now has a mandatory reference to a `Service S2`. The descriptor created for `Extender` does not have this mandatory reference and thus may cause unexpected runtime errors (probably `NullPointerException`).

Another problem with full inheritance support is that implementations have to be exported. For the extending classes to have access to the base classes, those must be available in the class space of the extending class. This requires components to be exported. But this violates a basic assumption of DS which deems it best practice to not expose implementation details through export.

For these two reasons it was decided at the Basel F2F to support inheritance for components within the same bundle.

As it is very hard for tooling – up to impossible – to decide whether a (parent) class is within the same bundle, it was decided at the Palo Alto F2F to drop inheritance completely.

### 8.3 Create separate Service annotation (Bug 2140)

The original bug 2140 asked for the creation of a separate service annotation. However, due to impedance mismatch on the default of `@Component.service()` and a new `@Service` annotation, it was decided after the F2F meeting in Palo Alto to drop this additional annotation.

### 8.4 Component provided service properties (Bug 2250)

Different ways of supporting changes of service properties through the component have been discussed like returning a map from the activation method and/or having a setter method on the component context. However this would create several problems like how to update the properties and when especially with factory components. Therefore it was decided at the Palo Alto F2F to drop this enhancement.

### 8.5 Create separate Property annotation (Bug 2141)

The original bug 2141 asked for a separate property annotation. After extensive considerations this suggestion has been withdrawn and replaced with the new annotation based approach to define properties.

### 8.6 New Life Cycle States

The following proposed new states were removed per bug 2567.

Two additional states describing the life cycle of a component are added: activating and deactivating. A component is in the activating state when it is leaving the satisfied state while it is activated by calling the activate method of a component. This state is a transient state and an immediate component enters the active state once it's activated and all other components enter the registered state. If activation fails, the component is back in the unsatisfied state.

If a component is deactivated, it enters the deactivating state during this process. Once deactivation is finished it is disposed.
9 Security Considerations

The diagnostic API has security implications in that it allows to introspect into component declarations and instances which are otherwise not accessible. In addition the API provides functionality to actually disable or enable components, although this is only temporary and reverted by a system or bundle restart.

Thus the complete API should only be available to management agents. Since this is a simple have-it-or-not situation, any bundle requiring access to the diagnostic API must have the ServicePermission[ServiceComponentRuntime, GET] permission.

10 Document Support

10.1 References

[3] Data Transfer Objects, Core Release 6

10.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Felix Meschberger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Adobe Systems Incorporated</td>
</tr>
<tr>
<td>Address</td>
<td>Barfüsserplatz 6, 4055 Basel, Switzerland</td>
</tr>
<tr>
<td>Voice</td>
<td>+41 61 226 55 49</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:fmeschbe@adobe.com">fmeschbe@adobe.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Carsten Ziegeler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Adobe Systems Incorporated</td>
</tr>
<tr>
<td>Address</td>
<td>Barfüsserplatz 6, 4055 Basel, Switzerland</td>
</tr>
<tr>
<td>Voice</td>
<td>+41 61 226 55 0</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:cziegele@adobe.com">cziegele@adobe.com</a></td>
</tr>
</tbody>
</table>
10.3 Acronyms and Abbreviations

10.4 End of Document
Abstract

While OSGi services are very powerful, some still find it challenging to use them effectively. This RFC looks at how CDI can be used to interact with the OSGi service layer. The intent is to bring the popular CDI programming model to OSGi as a way to interact with OSGi services. It will provide the convenience of CDI and allows developers familiar with the CDI technology to reuse their skills in an OSGi context.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

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

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

<table>
<thead>
<tr>
<th>0 Document Information</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 License</td>
<td>2</td>
</tr>
<tr>
<td>0.2 Trademarks</td>
<td>3</td>
</tr>
<tr>
<td>0.3 Feedback</td>
<td>3</td>
</tr>
<tr>
<td>0.4 Table of Contents</td>
<td>3</td>
</tr>
<tr>
<td>0.5 Terminology and Document Conventions</td>
<td>4</td>
</tr>
<tr>
<td>0.6 Revision History</td>
<td>4</td>
</tr>
</tbody>
</table>

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

2 Application Domain ........................................................................................................... 6
   2.1 CDI.............................................................................................................................. 6
   2.1.1 Example                                                                                     ................................................ 7
   2.2 Weld-OSGi.......................................................................................................................... 7
   2.2.1 Weld-OSGi example........................................................................................................ 8
   2.3 Declarative Services, Blueprint and CDI............................................................................... 8
   2.4 Terminology + Abbreviations................................................................................................ 8

3 Problem Description ........................................................................................................... 9

4 Requirements ...................................................................................................................... 9
   4.1 Functional Requirements.................................................................................................... 9
   4.2 Non-functional Requirements.......................................................................................... 11
   4.3 Requirements from RFP 98 (OSGi/Java EE umbrella RFP).................................................. 11
Technical Solution

5.1 Entities
5.2 CDI Container Life-Cycle
5.3 Requirements and Capabilities
5.4 Managed Beans and OSGi Services
   5.4.1 Publishing plain CDI providers in the Service Registry
5.5 Scopes for beans annotated as @Component
5.6 Life-Cycle
   5.6.1 Required dependencies (@Component only)
   5.6.2 Optional dependencies
   5.6.3 Allowing Service Injections in vanilla CDI injection points
   5.6.4 Service and bundle registration observers
   5.6.5 Service Filters
5.7 BundleContext injection
5.8 CdContainerBeanManager service
5.9 osgi.extender Capability
5.10 osgi.contract Capability

Data Transfer Objects

Javadoc

Considered Alternatives

Security Considerations

Document Support

References

Author’s Address

Acronyms and Abbreviations

End of Document

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

0.6 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>16/10/12</td>
<td>Initial version of the draft. David Bosschaert <a href="mailto:david@redhat.com">david@redhat.com</a></td>
</tr>
</tbody>
</table>
1 Introduction

While OSGi services are very powerful, consuming them has been a challenge for many OSGi users. There have been a number of solutions to this problem both in OSGi specifications as well as in non-standardized technologies. OSGi Declarative Services and Blueprint are popular specifications in this area, however they
provide new programming models that users need to learn. As of JavaEE 6, CDI (JSR 299) is included as a standard injection technology for JavaEE components. The CDI programming model seems suitable for interaction with the OSGi service layer as well and has the benefit that developers who are familiar with CDI don't need to learn a new technology in order to interact with the OSGi service registry.

This document proposes that OSGi will support CDI with the goal of creating a specification that describes how the CDI programming model can be used to interact with OSGi services.

2 Application Domain

Software developers often need to build loosely coupled applications. The need for this stems from a number of factors:

- Developing reusable services for consumption outside of the team
- Allowing those services to be easily consumed
- Unit testing of applications and services
- Allowing larger teams to work effectively together by isolating areas of development

Software developers also wish to using a standardized programming model. This promotes:

- Transferability of skill sets
- Ease of sourcing new developers and low initial overhead
- Clear understanding of correct behavior when unexpected behavior is encountered
- Consistency of programming model across the technological strata to provide a uniformity of approach to aid understanding

Finally, software developers require an environment in which the focus can be on solving business issues rather than technological issues. This allows a more responsive development process.

2.1 CDI

CDI, Contexts and Dependency Injection is specified by JSR 299. It defines a clean, mostly annotations-based injection model which has recently become very popular. CDI is part of JavaEE 6 but can also be used standalone in a JavaSE context.

Weld (http://seamframework.org/Weld) is the Reference Implementation of JSR 299.
2.1.1 Example

Although many advanced features are available, the most basic annotation used in CDI is `javax.inject.Inject` which declares the injection points for CDI.

For example the following Servlet class uses CDI injection to obtain an implementation of the `WeatherBean` interface.

```java
public class CDIServlet extends HttpServlet {
    @Inject
    WeatherBean weatherBean;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        PrintWriter writer = resp.getWriter();
        writer.print("The Weather in Amsterdam: "+
            weatherBean.getDescription("Amsterdam"));

        writer.flush();
        writer.close();
    }
}
```

While for the most basic use a CDI provider does not need to be annotated, CDI will attempt to find an implementor class and instantiate it using a no-arg constructor. Other mechanisms to publish a bean into CDI can be defined by using the `javax.enterprise.inject.Produces` annotation. Additionally, a number of scopes are defined that can be used to declare the lifecycle of a CDI bean.

For example, the `WeatherBean` above can be scoped to the application lifecycle by adding the `javax.enterprise.context.ApplicationScoped` annotation, as in this example:

```java
@ApplicationScoped
public class WeatherBeanProducer {
    @Produces @ApplicationScoped
    public WeatherBean newWeatherBean() {
        WeatherBean wb = new WeatherBeanImpl();
        wb.initialize();
        return wb;
    }

    public void disposeWeatherBean(@Disposes WeatherBean wb) {
        wb.cleanup();
    }
}
```

For more information see the CDI specification at JSR 299

2.2 Weld-OSGi

The Weld-OSGi project (http://mathieuancelin.github.com/weld-osgi/) has created an integration between CDI and OSGi. It allows CDI beans to be exposed as OSGi services and CDI injections to be satisfied by OSGi services. Weld-OSGi takes additional OSGi features into account such as service registration properties and the dynamic aspects of the Service Registry.
Furthermore, Weld-OSGi provides annotation based injection for the Bundle, BundleContext, Bundle Headers and the private bundle storage facility.

Additionally Weld-OSGi provides annotations-based integration with Service and Bundle events.

### 2.2.1 Weld-OSGi example

Many examples can be found in the weld-osgi documentation.

Weld-OSGi typically uses additional annotations to interact with the OSGi service Registry. For example, the `org.osgi.cdi.api.extension.annotation.Publish` annotation publishes the CDI bean in the OSGi Service Registry:

```java
@Publish
@ApplicationScoped
public class MyServiceImpl implements MyService {
    @Override
    public void doSomething() { ... }
}
```

To have a CDI injection come from the OSGi Service Registry, use the `OSGiService` annotation:

```java
@Inject @OSGiService MyService service;
```

OSGi Services can also be selected by using LDAP filters:

```java
@Inject @OSGiService @Filter("(&(lang=EN)(country=US))") MyService service;
```

For more examples, see the weld-osgi documentation.

### 2.3 Declarative Services, Blueprint and CDI

In Java EE, the EJB and CDI containers are able to collaborate such that EJB manages an EJB component's lifecycle, whilst CDI manages its runtime dependencies. For example, when a new EJB is created it can be handed over to the CDI container for it to process the injections (`@Inject`) before finally being made available for use. This relationship helps ensure a complementary positioning between the different component models and reduces runtime duplication (EJB is not required to handle `@Inject` processing itself).

OSGi has two existing component models in the form of Declarative Services and Blueprint. Each has its own mechanism for injection of services and Blueprint also supports bean injection within a bundle. Neither has standards support for runtime annotations for injection, although there is some Blueprint prototype work in Apache Aries. In addressing any requirements for runtime annotations support, serious consideration should be given to the use of existing annotations, such as `@Inject`. It also makes sense to consider creating similar complementary relationship between their containers and the CDI container for runtime injection processing, thus reducing duplication between various component model containers.

### 2.4 Terminology + Abbreviations

CDI – Context and Dependency Injection for JavaEE. Specified in JSR 299.
3 Problem Description

CDI provides a standardized, type-safe, loosely coupled programming model for Java EE 6 and above. Furthermore, it introduces powerful extensibility into the Java EE programming model, and promotes an ecosystem of “portable extensions”.

CDI is declarative, with metadata provided via annotations. This allows developers to locate all logic and metadata in a single location, allowing easier comprehension of the application.

CDI does not specify any modularity or inter-application communication, relying instead on the Java EE platform to provide this.

OSGi provides the de facto standard within Java for modular, service orientated programming.

Use of CDI in the context of OSGi provides a very compelling programming model. However, today there is no standard way to achieve this. A standard for leveraging CDI in OSGi will provide a migration path between JavaEE and OSGi where developers familiar with CDI can reuse their skills in both contexts without being locked in to a particular implementation.

4 Requirements

4.1 Functional Requirements

CDI001 – The specification MUST make it possible to use the CDI annotations and XML descriptor in an OSGi bundle to expose and consume CDI beans.

CDI003 – The specification MUST make it possible to consume OSGi services in CDI @Inject injection points in an OSGi bundle.

CDI004 – The specification MUST make it possible to select OSGi services used in CDI beans based on OSGi filters.

CDI005 – The specification MUST make it possible to consider CDI qualifiers when looking up CDI beans in the OSGi Service Registry.

CDI014 – The specification MUST provide a mechanism to specify additional OSGi service registration properties for CDI beans.

CDI006 – The specification MUST make it possible to write a portable CDI jar that runs both in JavaEE as well as in OSGi.
CDI007 – The specification MUST consider the thread-safety issues that can arise when migrating CDI beans from JavaEE to OSGi.

CDI008 – The specification MUST consider the issues that can arise in relation to the dynamic bundle lifecycle in OSGi.

CDI015 – The specification MUST consider the issues that can arise with OSGi service dynamism when these services are injected into a CDI bean.

CDI009 – The specification MUST make it possible to take advantage of the dynamic service capabilities of OSGi.

CDI016 – The specification MUST extend the life-cycle dependency model as provided in CDI, to support the dynamic life-cycle provided by OSGi. For example, it MUST NOT be fatal to deploy a CDI bean that does not have all its dependencies initially satisfied and it MUST be possible to change bean dependencies without requiring the CDI application to be redeployed or restarted.

CDI031 – The specification MUST extend the life-cycle dependency model of CDI to include dynamic OSGi service dependencies.

CDI017 – The specification MUST make it possible to declare a CDI injection point to an OSGi service as optional.

CDI018 – The specification MUST provide a mechanism to consume multiple matching services/beans of a given type in an injection point. For example via the @Inject Instance<T> mechanism.

CDI019 – The specification MUST support CDI events as defined by the CDI specification.

CDI021 – The specification MAY provide a deep integration between CDI events and OSGi events or other OSGi mechanism.

CDI020 – The specification MUST support CDI extensions as defined by the CDI specification.

CDI022 – the specification MAY provide a deep integration between CDI extensions and OSGi services or other OSGi mechanism.

CDI010 – The specification MAY introduce additional annotations.

CDI011 – The specification MUST define the behavior in case of incorrect CDI metadata.

CDI012 – The specification MUST NOT prevent the use of @Inject (and other common java annotations) in other component models/technologies present in the OSGi Framework.

CDI013 – The specification MUST define an opt-in mechanism. Bundles not opting in MUST not be considered by the CDI-OSGi integration layer.

CDI023 – All the inter-bundle interaction between CDI beans MUST go through the OSGi Service Registry.

CDI024 – The specification MUST make it possible to access the BundleContext from inside a CDI bean in an OSGi Framework.

CDI025 – The specification MUST provide support for @PostConstruct and @PreDestroy activation and de-activation callbacks.
CDI026 – The specification SHOULD consider defining behavior for relevant CDI scopes.

CDI027 – The solution MAY define new scopes for use with CDI inside an OSGi Framework.

CDI028 – The specification MUST define an opt-in mechanism for CDI extensions.

CDI029 – The specification MUST consider the issues that arise from dynamically adding CDI extensions to the system.

CDI030 – The specification MUST support the inclusion of CDI beans and descriptors in a Web Application Bundle in the same way they can be included in a WAR (e.g. including beans.xml in WEB-INF/).

CDI032 – The specification MUST support the OSGi Service Permission security model when publishing OSGi services from CDI beans and injecting services into CDI beans. It needs to take into account that the CDI extender acts on behalf of other bundles and uses the permissions associated with those.

### 4.2 Non-functional Requirements

CDI050 – The specification MUST NOT prevent an implementation from injecting OSGi services into CDI beans which are not deployed as OSGi bundles.

CDI052 – The specification MUST NOT prevent implementation from CDI050 – The specification MUST NOT prevent an implementation from injecting OSGi services into CDI beans which are not deployed as OSGi bundles.

CDI051 – The specification SHOULD adhere to the current CDI programming model as much as possible.

### 4.3 Requirements from RFP 98 (OSGi/Java EE umbrella RFP)

JEE001 – A Java EE/OSGi system SHOULD enable the standard Java EE application artifacts (e.g. web application) to remain installed when a supporting Java EE runtime element (e.g. web container) is dynamically replaced.

JEE002 – RFCs that refer to one or more Java EE technologies MUST NOT impede the ability of an OSGi-compliant implementation to also be compliant with the Java EE specification.

JEE003 – RFCs that refer to one or more Java EE technologies MAY define the additional aspects of the technology that are required for the technology to be properly integrated in an OSGi framework but MUST NOT make any syntactic changes to the Java interfaces defined by those Java EE specifications.

JEE004 – RFCs whose primary purpose is integration with Java EE technologies MUST NOT require an OSGi Execution Environment greater than that which satisfies only the signatures of those Java EE technologies.
5 Technical Solution

5.1 Entities

- CDI – Contexts and Dependency Injection 1.0 (JSR-299).
- CDI Provider – An implementation of the CDI 1.0 specification.
- CDI OSGi adapter – Adapts a given CDI Provider to the OSGi environment. This entity is implementation dependent and may or may not be separate from the CDI provider.
- CDI Bundle – A CDI-enabled OSGi bundle.
- CDI Container – A container for managed beans in a CDI Bundle. Each CDI Bundle has its own CDI container.
- CDI Extender – An application of the extender pattern to discover CDI Bundles and to manage the CDI container life-cycle on behalf of CDI Bundles.
- CDI Extension – A portable extension as defined in CDI 1.0.
- Extension Bundle – A bundle providing one or more CDI extensions. An extension bundle may or may not be a CDI Bundle at the same time.
- OSGi CDI Extension – A specific CDI extension for publishing and consuming OSGi services to or from managed beans by means of annotations is is a mandatory part of this specification.

5.2 CDI Container Life-Cycle

The CDI Extender tracks all bundles becoming ACTIVE. When a tracked bundle is identified as a CDI Bundle, the CDI Extender creates a CDI Container for this bundle. When a tracked CDI Bundle is stopped, the CDI Extender stops the CDI Container for the given bundle.

Starting a CDI container requires scanning the CDI Bundle for managed bean candidate classes. Class loading scenarios are far more complex in OSGi than in Java EE or Java SE, due to the modular and dynamic nature of the OSGi environment.

The bean scanner needs to consider

All classes on the bundle classpath of the CDI Bundle, including any embedded archives or directories and fragments as well as all classes visible via imported package or bundle wirings. This maps to the resources returned via the BundleWiring.listResources("/", ".class", 0) method call.

The set of candidate bean classes determined by the bean scanner is not equal to the set of managed beans in the bean container. The CDI Provider discards all candidate classes that do not satisfy the requirements for managed beans. The set of managed beans may be further extended or modified by CDI extensions.
5.3 Requirements and Capabilities

Bean deployment archives according to CDI 1.0 are required to opt in to OSGi enrichment by the CDI Extender. However, opting in may have no effect at all if a would-be CDI Bundle is installed and started in a system where no CDI Extender is available.

This kind of dependency can be made explicit using capabilities (introduced in OSGi Core 4.3). Capabilities are also useful to express a loose dependency on a given CDI extension. This covers the following use case:

CDI Bundle A works with a CDI extension with an annotation API defined in bundle B and an extension implementation in bundle C. A has a package dependency on B, but not on C. There is only an implicit runtime dependency on C. C needs to be resolved when the CDI container for A is constructed, so that the extension can modify the set of managed beans, but C may not be available at all.

Extension capabilities are a means to declare a dependency on a given extension by name and not by implementation.

For these reasons, this specification defines the following capabilities.

- An OSGi extender capability named osgi.cdi.
- A capability osgi.cdi.extension with a mandatory attribute extension = <name>, where <name> is a logical name for the given extension. The name osgi.cdi is reserved for the OSGi CDI extension defined in this specification.

A CDI Bundle MUST require the OSGi extender capability named osgi.cdi, e.g.

```
Require-Capability: osgi.extender; filter:="(osgi.extender=osgi.cdi)"
```

(Note that this is not a sufficient condition for a bundle to be a CDI Bundle. A CDI Bundle must also be a bean deployment archive as defined in CDI 1.0, i.e. it must have a beans.xml descriptor in one of the defined locations.)

A CDI Bundle MAY require additional osgi.cdi.extension capabilities with other filter attributes for other CDI extensions.

A CDI Extender implementation MUST provide the osgi.extender capability named osgi.cdi with version 1.0, e.g.

```
Provide-Capability = osgi.extender; osgi.extender=osgi.cdi; version=1.0
```

Future versions of this specification will require higher version numbers.

An OSGi-enabled CDI extension MUST provide an osgi.cdi.extension capability with a distinctive name, e.g.

```
Provide-Capability = osgi.cdi.extension; extension=frobnicator
```

5.4 Managed Beans and OSGi Services
A class can be published to the OSGi service registry using the @org.osgi.service.cdi.Component annotation. Note that classes that are already CDI beans, are not automatically published to the service registry; the @Component annotation or metadata as described in section 5.4.1 is always required for this. The requirement to explicitly annotate a bean to be an OSGi component is necessary for the following reasons:

1. In CDI 1.0 every class is a potential CDI bean, no annotations are required for this. The container scans injection points at container startup time to calculate which classes should be registered as beans. A CDI container in OSGi is scoped to a single bundle, and will only know about injection points in that bundle. OSGi services however can be used by other bundles as well. Although it is technically possible to find injection points in all bundles, this would be very hard to reason about for end users.

2. Not each CDI bean should be registered to the service registry. CDI is often used to inject tightly coupled classes into each other. This is find for internal bundle usage, but should not be reflected in the service registry.

3. Not all CDI beans might be exported. It's useless to publish services of types that other bundles can't have access to.

This behavior is different then the behavior described in the EJB integration specification, where all EJBs are published as OSGi services by default. Although technically EJBs and CDI beans are very similar, their usage in practice is often very different. EJBs tend to be used in a similar granularity as OSGi services, while CDI beans are not.
An example @Component annotation:

```java
@Component(interfaces={A.class})
public class MyComponent implements A,B {}
```

The `interfaces` annotation element is optional. If not specified the component is registered in the OSGi Service Registry under all the interfaces it directly implements. If the component does not implement any interfaces it will be registered under its implementation class.

Service properties can be defined in the `@Component` annotation using the “properties” argument. The properties argument takes an array of `@ComponentProperty` annotations. The `@ComponentProperty` annotations requires a key and value to be set, both of type `String`. An optional type element can be used to specify the data type of the property value. Supported data types are: service property values may only be one of the following types:

- **Scalar** – String, Integer, Long, Float, Double, Byte, Short, Character, Boolean.
- **Array** – An array of the allowable scalar types, declared using the following syntax: `String[]` or `Float[]`.
- **Collection** – A list or set that contains scalar types declared using the following syntax: `List<String>` or `Set<Short>`.

```java
@Component(properties = {@ComponentProperty(key = "key", value = "value"), @ComponentProperty(key = "key2", value = "42", type = "long")})
public class ExampleComponent {}
```

### 5.4.1 Publishing plain CDI providers in the Service Registry

To support integration of existing CDI provider classes with OSGi an alternative to the `@Component` annotation is available via the OSGi bundle manifest. This allows beans that act as providers in plain CDI environments to participate in the OSGi integration as services without having to change and recompile these components. A class can be declared as a component in the manifest through the `component` directive on the `osgi.extender=osgi.cdi` requirement:

```text
Require-Capability: osgi.extender;
   filter:="(osgi.extender=osgi.cdi)";
   components:="org.acme.MyComponent,org.acme.ExampleComponent"
```

### 5.5 Scopes for beans annotated as @Component

The following table outlines the CDI scopes and their support by this specification:

<table>
<thead>
<tr>
<th>Scope Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ApplicationScoped</td>
<td>Same as @Singleton.</td>
</tr>
<tr>
<td>@ConversationScoped</td>
<td>Not yet supported. Normally used in the context of JSF.</td>
</tr>
<tr>
<td>@Dependent</td>
<td>Default, map to @SingletonPrototypeScoped</td>
</tr>
<tr>
<td>@SessionScoped</td>
<td>Not supported yet in OSGi (web request specific)</td>
</tr>
<tr>
<td>@Singleton</td>
<td>Same as @SingletonScoped</td>
</tr>
<tr>
<td>@RequestScoped</td>
<td>Not supported yet in OSGi (web request specific)</td>
</tr>
</tbody>
</table>

The following table lists OSGi-defined scopes and associated behavior. These scopes are only relevant for CDI beans registered in the OSGi Service Registry:

<table>
<thead>
<tr>
<th>Scope Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@BundleScoped</td>
<td>Maps to OSGi Service Factory</td>
</tr>
</tbody>
</table>
5.6 Life-Cycle

Inject OSGi services in CDI beans

The life-cycle of an @Component is different from a normal CDI bean, because it has to deal with service dynamics. An @Component may have dependencies on other OSGi services which can influence the life-cycle of the @Component as described in the next section.

An OSGi service can be injected into another OSGi Service or a CDI bean using the standard @Inject annotation. The life-cycle of an OSGi service is different than a plain CDI bean however. By default, a CDI bean has the Dependent scope; it's life-cycle is bound to the life-cycle of the consuming bean, or alternatively it's scope is defined using annotations. For a mapping of CDI scope to OSGi behaviour, see section 5.5, however not all CDI scopes are supported in an OSGi environment. Another difference is that there may be multiple OSGi services publishing the same interface, a client may or may not choose a specific instance using service properties and ordering.

Because of the differences between the CDI BeanManager and OSGi service registry and to remove ambiguity we must always use @Service CDI qualifier to instruct the CDI container (and the developer) that we are dealing with a specific type of bean, where different rules may apply. When the @Service annotation is used, the CDI container will lookup the OSGi service in the service registry and inject it's instance into the CDI bean.

Any OSGi service registered in the service registry can be injected using CDI, even if the service was not registered using CDI but with the low level service API or DS for example.
5.6.1 Required dependencies (@Component only)

Require a dependency; the component will not be registered when a required dependency is not available. This is the default behavior. When all dependencies becomes available, the component will be registered. If at some point during runtime a required dependency becomes unavailable, the component will be deregistered again. This model handles service dynamics correctly, while the code doesn't have to handle the case where dependencies are not available.

As an example we have the dependency structure:

When C becomes unavailable, B will become unavailable as well, and so will A. When C becomes available again, so will be B, and so will A.

5.6.2 Optional dependencies

To deal with service dynamics a proxy is injected instead of the real reference to the service. The proxy is by default a null-object, every method invocation should return null. The code that uses an optional dependency must deal with the possibility that the service is not available and handle null values properly.

Alternatively the @Service supports a configuration parameter “proxyType” that can be used to configure the proxy to throw a org.osgi.cdi.ServiceNotAvailableException when the proxy invoked while the service is not available. This way clients can handle null values returned by the real service differently than the situation where the service is not available.

Optional and required dependencies can be mixed in a single component and in the dependency graph of a component. Take the following example again, note that the dependency on C is now optional.

When C becomes unavailable, B should still be available, and therefore A as well. When B becomes unavailable, A should still become unavailable as well. Because the dependency on C is now optional, the code of B should be handling the fact that method invocations on C might return null or throw exceptions.

CDI does not support dynamic dependencies. All beans must be registered and all dependencies must be resolved at container startup. For normal CDI beans the container should resolve @Inject @Service injection points immediately with proxies as described for optional dependencies. Required dependencies are not required to be implemented by the container for normal CDI beans; the container may throw an exception to inform the developer that the required attribute is not supported. A CDI container may implement required dependencies for normal CDI beans as well too however.

For components registered with the @Component annotation, required dependencies must be supported. An @Component with unavailable dependencies must not be registered to the OSGi service registry, and it's @PostConstruct method must not be invoked. If the component was active before the dependency became unavailable, the component is deregistered when the required dependency reappears.
unavailable, the @PreDestory method must be called and the service must be de-registered from the service registry.

Because OSGi services are dynamic, a developer should make the explicit choice to inject beans using the OSGi service registry by using the @Service qualifier. If the @Service qualifier is not used, the container should not query the service registry.

5.6.3 Allowing Service Injections in vanilla CDI injection points

As described in previous sections, the life-cycle and dynamics of OSGi makes this a fundamentally different environment to the JavaEE container for which CDI was originally designed. This is one of the key reasons for introducing the @Service annotation to go with the @Inject annotation.

However in certain scenarios it may be desirable allow injections from the OSGi service registry into plain @Inject injection points. This is to allow migration paths from pure JavaEE to configurations where unmodified CDI beans are injected with services from the OSGi service registry. Clearly such a situations introduce risks and analysis of the system is necessary beforehand to ensure that no fatal situations can occur by the introduced dynamics.

Once satisfied that the system can cope in the OSGi environment, injection into pure @Inject points can be enabled by setting a directive on the extender requirement in the CDI Bundle:

```
Require-Capability: osgi.extender;
    filter:="(osgi.extender=osgi.cdi)";
    at-service:="optional"
```

5.6.4 Service and bundle registration observers

In most situations service dependencies are injected directly into a field. Sometimes some extra code needs to be executed however. This can be done using callback methods. There is a callback method for service registration and a callback for service deregistration. The parameters of the observer method should be the type of the service.

Only events fired while the bundle containing the observer methods are delivered. Events fired while the bundle was not active are ignored.

```java
void serviceAdded(@Observes @ServiceAdded SomeService) ;
void serviceRemoved(@Observes @ServiceRemoved SomeService) ;
void serviceModified(@Observes @ServiceModified SomeService) ;
```

In some cases it’s useful to also have access to the ServiceReference representing a service. For this case a special event type is introduced:

```java
public class ServiceCdiEvent<T> {
    private ServiceReference<T> reference;
    private T service;
    // constructor and getters etc.
}
```

```java
void serviceAdded(@Observes @ServiceAdded ServiceCdiEvent<SomeService> event)
void serviceRemoved(@Observes @ServiceRemoved ServiceCdiEvent<SomeService> event)
```

The @ServiceAdded and @ServiceRemoved annotations have the same service filtering semantics as @Service described in the following section.
5.6.5 Service Filters

Service filtering can be done using the @Service value parameter using the standard OSGi service filter (LDAP like) syntax:

@Service("(somekey=somevalue)")

As a convenience alternatively the same could be done using CDI Qualifiers:

@Qualifier
@ServiceFilter

public @interface SomeKey {
    String value();
}

@Inject @Service @SomeKey("somevalue")

MyService service;

The name of the qualifier is used as the name of the property, the value passed into the qualifier annotation the value. If the value is not a String (e.g. an enum value), the String representation of the value is used. If the qualifier has a default value, the annotation can be used without specifying a value. In the following example the used filter is "(somekey=somevalue)". This mechanism provides a slightly more type-safe approach to using service filters.

@Qualifier
@ServiceFilter

public @interface SomeKey {
    String value() default "somevalue";
}

@Inject @Service @SomeKey

MyService service;

It is possible to combine multiple qualifiers to create AND filters. In the following example both annotations are qualifiers, and the resulting filter is "(&{(somekey=somevalue)(someotherkey=someothervalue)})".

@Inject @Service @SomeKey("somevalue") @SomeOtherKey("someothervalue")

MyService service;

More complex filters are not supported using qualifiers, the standard filter syntax should be used instead with the @Service annotation.

When using OSGi service filters it is common to use dots in property names, e.g. 'service.ranking'. It is not possible however to use dots in a Java Annotation type name. To work around this issue the Annotation
type name is parsed and certain markers in the name are interpreted as dots. Because service
properties are case insensitive, camel casing can be used as a marker.

For example the qualifier `@MyServiceProperty("example")` translates to 
"(my.service.property=example)". The first
character is lowercased. Each following capital is translated to a dot.

Requiring configuration

An `@Component` can require a Config Admin Configuration object with a specific PID. The component will only
become available when a configuration object with the specified PID is found. This is useful when a component
does not have usable default configuration values. If the configuration object is not available, the behaviour is the
same as for a unavailable required service dependency.

A configuration dependency can be configured as follows:

```java
@Component(requireConfiguration="PID.of.configuration")
```

### 5.7 BundleContext injection

It's possible to inject the BundleContext using CDI annotations.

```java
@Inject BundleContext context;
```

This will always inject the BundleContext of the bundle that registered the component, even if the component's
class was imported from another bundle. For example we could have the following scenario. Although the `@Inject
BundleContext` annotation is declared in a class exported by Bundle A, Bundle B does the actual component
registration. This means that the BundleContext of bundle B will be injected.

Bundle A

```java
public abstract class MyBaseClass {
    @Inject BundleContext bundleContext;
}
```

Bundle B

```java
@Component
public class MySubClass extends MyBaseClass {
}
```

### 5.8 CdiContainerBeanManager service

The implementation will provide a `CdiContainer` service which provides access to a registered under
the `javax.enterprise.inject.spi.BeanManager` interface implementation. The BeanManager provides a
standard portable introspective interfaces into the CDI container.
The BeanManager service is registered under the Bundle Context of the associated CDI bundle, as each CDI bundle has its own container. As a result many CdiContainer services will be present in the system.

### 5.9 osgi.extender Capability

The implementation must provide an osgi.extender capability as follows:

```text
Provide-Capability: osgi.extender;
    osgi.extender="osgi.cdi";
    uses:="org.osgi.service.cdi, javax.enterprise.inject.spi";
    version:Version="1.0"
```

### 5.10 osgi.contract Capability

The OSGi Enterprise specification version 5 defines the osgi.contract capability namespace and RFC 180 defines mappings of JSR-defined technologies to these capabilities. Relevant technologies from RFC 180 to this specification are the JavaCDI and JavaInject API contracts.

An implementation of this specification is not required to export the associated `javax.*` packages, but if it does, it must also provide the JavaCDI and JavaInject capabilities in the osgi.contract namespace.

---

### 6 Data Transfer Objects

RFC 185 defines Data Transfer Objects as a generic means for management solutions to interact with runtime entities in an OSGi Framework. DTOs provides a common, easily serializable representation of the technology.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a DTO? The expectation is that in most cases it would.

The DTOs for the design in this RFC should be described here and if there are no DTOs being defined 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.

---

### 7 Javadoc

---

Copyright © OSGi Alliance 2013 All Rights Reserved
### Package Summary

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.service.cdi</td>
<td>CDI Integration Package Version 1.0.</td>
<td>23</td>
</tr>
</tbody>
</table>
Package org.osgi.service.cdi

CDI Integration Package Version 1.0.

See:

Description

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CdiContainer</td>
<td>A CdiContainer object is registered by the cdi extender in the OSGi registry for each managed CDI bundle.</td>
</tr>
<tr>
<td>CdiListener</td>
<td>A CdiListener object can be registered in the OSGi registry.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CdiEvent</td>
<td>CdIEvent are sent by the cdi extender and received by registered CdiListener services.</td>
</tr>
<tr>
<td>ComponentEvent</td>
<td>CDI Event sent by the CDI extender whenever a component has been registered or unregistered to/from the OSGi registry.</td>
</tr>
<tr>
<td>Constants</td>
<td>Defines CDI constants</td>
</tr>
<tr>
<td>ServiceEvent</td>
<td>The event sent by the CDI extender whenever a service that matches an injection point is registered or unregistered from the OSGi registry.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enum Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceType</td>
<td>Defines the type of service injected.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exception Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNotAvailableException</td>
<td>Exception that can be thrown from the injected Service if a matching service is not available.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Annotation Types Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>The @Component annotation exposes a CDI bean in the OSGi registry.</td>
</tr>
<tr>
<td>ComponentProperty</td>
<td>Annotation used to define a property that will be associated to Component registered in the OSGi registry.</td>
</tr>
<tr>
<td>ComponentRegistered</td>
<td>The @ComponentRegistered can be used to observe ComponentEvent events whenever a component has been registered from the OSGi registry.</td>
</tr>
<tr>
<td>ComponentUnregistered</td>
<td>The @ComponentUnregistered can be used to observe ComponentEvent events whenever a component has been unregistered from the OSGi registry.</td>
</tr>
<tr>
<td>Filter</td>
<td>The @Filter annotation can be used to filter objects.</td>
</tr>
<tr>
<td>Service</td>
<td>The @Service annotation can be used to annotate a CDI injection point and inform the CDI extender that the injection should be done by a service grabbed from the OSGi registry.</td>
</tr>
<tr>
<td>ServiceAdded</td>
<td>The @ServiceAdded can be used to observe ServiceEvent events whenever a matching service injected via @Service is registered in the OSGi registry.</td>
</tr>
<tr>
<td>ServiceRemoved</td>
<td>The @ServiceRemoved can be used to observe ServiceEvent events whenever a matching service injected via @Service is unregistered in the OSGi registry.</td>
</tr>
</tbody>
</table>

Package org.osgi.service.cdi Description

CDI Integration 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.cdi; version="[1.0,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.service.cdi; version="[1.0,1.1)"
public interface CdiContainer

A CdiContainer object is registered by the the cdi extender in the OSGi registry for each managed CDI bundle.

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeanManager getBeanManager()</td>
<td>25</td>
</tr>
</tbody>
</table>

**Method Detail**

*getBeanManager*

BeanManager getBeanManager()
Class CdiEvent

org.osgi.service.cdi

java.lang.Object

org.osgi.service.cdi.CdiEvent

class CdiEvent
extends Object

cdiEvent are sent by the cdi extender and received by registered CdiListener services.

See Also:
CdiListener

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int CREATED</td>
<td>26</td>
</tr>
<tr>
<td>static int CREATING</td>
<td>26</td>
</tr>
<tr>
<td>static int DESTROYED</td>
<td>27</td>
</tr>
<tr>
<td>static int DESTROYING</td>
<td>27</td>
</tr>
<tr>
<td>static int FAILURE</td>
<td>27</td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CdiEvent(int type, org.osgi.framework.Bundle bundle, org.osgi.framework.Bundle extenderBundle)</td>
<td>27</td>
</tr>
<tr>
<td>CdiEvent(int type, org.osgi.framework.Bundle bundle, org.osgi.framework.Bundle extenderBundle, Throwable cause)</td>
<td>27</td>
</tr>
<tr>
<td>CdiEvent(CdiEvent event, boolean replay)</td>
<td>27</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.framework.Bundle getBundle()</td>
<td>27</td>
</tr>
<tr>
<td>Throwable getCause()</td>
<td>28</td>
</tr>
<tr>
<td>org.osgi.framework.Bundle getExtenderBundle()</td>
<td>28</td>
</tr>
<tr>
<td>long getTimestamp()</td>
<td>27</td>
</tr>
<tr>
<td>int getType()</td>
<td>27</td>
</tr>
<tr>
<td>boolean isReplay()</td>
<td>28</td>
</tr>
</tbody>
</table>

Field Detail

CREATING

public static final int CREATING = 1

CREATED

public static final int CREATED = 2
Class CdiEvent

DESTROYING

public static final int DESTROYING = 3

DESTROYED

public static final int DESTROYED = 4

FAILURE

public static final int FAILURE = 5

Constructor Detail

CdiEvent

public CdiEvent(int type,
        org.osgi.framework.Bundle bundle,
        org.osgi.framework.Bundle extenderBundle)

CdiEvent

public CdiEvent(int type,
        org.osgi.framework.Bundle bundle,
        org.osgi.framework.Bundle extenderBundle,
        Throwable cause)

CdiEvent

public CdiEvent(CdiEvent event,
        boolean replay)

Method Detail

getType

public int getType()

getTimestamp

public long getTimestamp()

getBundle

public org.osgi.framework.Bundle getBundle()
Class CdiEvent

getExtenderBundle

public org.osgi.framework.Bundle getExtenderBundle()

getCause

public Throwable getCause()

isReplay

public boolean isReplay()
public interface CdiListener

A CdiListener object can be registered in the OSGi registry. The cdi extender will call each listener whenever an event happens.

See Also:
CdiEvent

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>void cdiEvent(CdiEvent event)</td>
<td>29</td>
</tr>
</tbody>
</table>

Method Detail

cdiEvent

void cdiEvent(CdiEvent event)
The @Component annotation exposes a CDI bean in the OSGi registry.

### Required Element Summary

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class&lt;?&gt;[]</td>
<td>interfaces</td>
<td>30</td>
</tr>
<tr>
<td>ComponentProperty[]</td>
<td>properties</td>
<td>30</td>
</tr>
<tr>
<td>String</td>
<td>requireConfiguration</td>
<td>30</td>
</tr>
</tbody>
</table>

### Element Detail

#### interfaces

```java
public abstract Class<?>[] interfaces
```

**Default:**

```java
{}
```

#### properties

```java
public abstract ComponentProperty[] properties
```

**Default:**

```java
{}
```

#### requireConfiguration

```java
public abstract String requireConfiguration
```

**Default:**

```java

```
public class ComponentEvent
extends Object

CDI Event sent by the CDI extender whenever a component has been registered or unregistered to/from the OSGi registry.

See Also:  
ComponentRegistered, ComponentUnregistered, javax.enterprise.event.Observes

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComponentEvent(org.osgi.framework.ServiceRegistration&lt;T&gt; registration, T service)</td>
<td>31</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getRegistration()</td>
<td>31</td>
</tr>
<tr>
<td>getService()</td>
<td>31</td>
</tr>
</tbody>
</table>

Constructor Detail

ComponentEvent

public ComponentEvent(org.osgi.framework.ServiceRegistration<T> registration, T service)

Method Detail

getRegistration

public org.osgi.framework.ServiceRegistration<T> getRegistration()  

ggetService

public T getService()
Annotation Type ComponentProperty

@Retention(value=RetentionPolicy.RUNTIME)
public @interface ComponentProperty

Annotation used to define a property that will be associated to Component registered in the OSGi registry.

<table>
<thead>
<tr>
<th>Required Element Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String key</td>
<td>32</td>
</tr>
<tr>
<td>String value</td>
<td>32</td>
</tr>
</tbody>
</table>

Element Detail

key

class ComponentProperty
{
    @Retention(value=RetentionPolicy.RUNTIME)
    @interface ComponentProperty

    // Annotation used to define a property that will be associated to Component registered in the OSGi registry.
}

    public abstract String key;

value

class ComponentProperty
{
    @Retention(value=RetentionPolicy.RUNTIME)
    @interface ComponentProperty

    // Annotation used to define a property that will be associated to Component registered in the OSGi registry.
}

    public abstract String value;
@Target(value=ElementType.PARAMETER)
@Retention(value=RetentionPolicy.RUNTIME)
@Documented
public @interface ComponentRegistered

The @ComponentRegistered can be used to observe ComponentEvent events whenever a component has been registered from the OSGi registry.

See Also: ComponentEvent, javax.enterprise.event.Observes
The `@ComponentUnregistered` can be used to observe `ComponentEvent` events whenever a component has been unregistered from the OSGi registry.

See Also:

- `ComponentEvent`
- `javax.enterprise.event.Observes`
Class Constants

org.osgi.service.cdi

java.lang.Object

_ org.osgi.service.cdi.Constants

final public class Constants
extends Object

Defines CDI constants

Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static String CDI_EXTENDER</td>
<td>String</td>
<td>Opt-in manifest header, starting a CDI container</td>
<td>35</td>
</tr>
<tr>
<td>static String CDI_EXTENSION_CAPABILITY</td>
<td>String</td>
<td>CDI_EXTENSION_CAPABILITY = &quot;osgi.cdi.extension&quot;</td>
<td>35</td>
</tr>
<tr>
<td>static String EXTENDER_CAPABILITY</td>
<td>String</td>
<td>EXTENDER_CAPABILITY = &quot;osgi.extender&quot;</td>
<td>35</td>
</tr>
</tbody>
</table>

Field Detail

CDI_EXTENDER

public static final String CDI_EXTENDER = "osgi.cdi"

Opt-in manifest header, starting a CDI container

EXTENDER_CAPABILITY

public static final String EXTENDER_CAPABILITY = "osgi.extender"

CDI_EXTENSION_CAPABILITY

public static final String CDI_EXTENSION_CAPABILITY = "osgi.cdi.extension"
Annotation Type Filter

org.osgi.service.cdi

```java
@Target(value={
    ElementType.FIELD,
    ElementType.TYPE,
    ElementType.PARAMETER,
    ElementType.CONSTRUCTOR
})
@Retention(value=RetentionPolicy.RUNTIME)
@Documented
public @interface Filter
```

The `@Filter` annotation can be used to filter objects. It can be applied to `@Service` or `@EventAdmin` injection points.

See Also:

- Service
- EventAdmin

### Required Element Summary

<table>
<thead>
<tr>
<th>Element</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String value</td>
<td>36</td>
</tr>
</tbody>
</table>

### Element Detail

**value**

```java
public abstract String value
```

**Default:**

````
````
### Annotation Type Service

`org.osgi.service.cdi`

```java
@Qualifier
@Target(value={
    ElementType.FIELD,
    ElementType.PARAMETER,
    ElementType.CONSTRUCTOR
})
@Retention(value=RetentionPolicy.RUNTIME)
@Documented
public @interface Service
```

The `@Service` annotation can be used to annotate a CDI injection point and inform the CDI extender that the injection should be done by a service grabbed from the OSGi registry.

**See Also:**
- Filter

<table>
<thead>
<tr>
<th>Required Element Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean <code>required</code></td>
<td>37</td>
</tr>
<tr>
<td><code>ServiceType</code> <code>type</code></td>
<td>37</td>
</tr>
</tbody>
</table>

#### Element Detail

**required**

```java
public abstract boolean required
```

**Default:**
- `false`

**type**

```java
public abstract `ServiceType` type
```

**Default:**
- `ServiceType.NullObject`
### Annotation Type ServiceAdded

```java
@Target(value=ElementType.PARAMETER)
@Retention(value=RetentionPolicy.RUNTIME)
@Documented
public @interface ServiceAdded
```

The `@ServiceAdded` can be used to observe `ServiceEvent` events whenever a matching service injected via `@Service` is registered in the OSGi registry.

See Also:  
[ServiceEvent](#), [javax.enterprise.event.Observes](#)
Class ServiceEvent
org.osgi.service.cdi

java.lang.Object
  org.osgi.service.cdi.ServiceEvent

public class ServiceEvent
extends Object

The event sent by the CDI extender whenever a service that matches an injection point is registered or unregistered from the OSGi registry.

See Also:
  ServiceAdded, ServiceRemoved, javax.enterprise.event.Observes

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceEvent(org.osgi.framework.ServiceReference&lt;T&gt; reference, T service)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>getReference()</td>
</tr>
<tr>
<td>getService()</td>
</tr>
</tbody>
</table>

Constructor Detail

ServiceEvent

public ServiceEvent(org.osgi.framework.ServiceReference<T> reference, T service)

Method Detail

getReference

public org.osgi.framework.ServiceReference<T> getReference()

getService

public T getService()
Class ServiceNotAvailableException

org.osgi.service.cdi

java.lang.Object
   | java.lang.Throwable
      | java.lang.Exception
         | java.lang.RuntimeException
            | org.osgi.service.cdi.ServiceNotAvailableException

All Implemented Interfaces:
   Serializable

public class ServiceNotAvailableException
extends RuntimeException

Exception that can be thrown from the injected Service if a matching service is not available.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNotAvailableException(String message)</td>
<td>40</td>
</tr>
</tbody>
</table>

Constructor Detail

ServiceNotAvailableException

public ServiceNotAvailableException(String message)
Annotation Type ServiceRemoved

`org.osgi.service.cdi`

```java
@Target(value=ElementType.PARAMETER)
@Retention(value=RetentionPolicy.RUNTIME)
@Documented
public @interface ServiceRemoved
```

The `@ServiceRemoved` can be used to observe `ServiceEvent` events whenever a matching service injected via `@Service` is unregistered in the OSGi registry.

See Also:
- `ServiceEvent`, `javax.enterprise.event.Observes`
**Enum ServiceType**

```java
org.osgi.service.cdi
```

java.lang.Object

```
java.lang.Enum<ServiceType>
```

org.osgi.service.cdi.ServiceType

All Implemented Interfaces:

`Comparable<ServiceType>`, `Serializable`

```java
public enum ServiceType
```

extends `Enum<ServiceType>`

Defines the type of service injected.

---

### Enum Constant Summary

<table>
<thead>
<tr>
<th>NullObject</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A dummy proxy that simply returns <code>null</code> when a matching OSGi service is not available.</td>
<td>42</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>valueOf(String name)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static <code>ServiceType</code></td>
<td>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>values()</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static <code>ServiceType[]</code></td>
<td>42</td>
</tr>
</tbody>
</table>

---

### Enum Constant Detail

**NullObject**

```java
public static final ServiceType NullObject
```

A dummy proxy that simply returns `null` when a matching OSGi service is not available.

---

### Method Detail

**values**

```java
public static `ServiceType[]` values()
```

**valueOf**

```java
public static `ServiceType` valueOf(String name)
```

---

Java API documentation generated with [DocFlex/Doclet v1.5.6](https://www.docflex.com)

DocFlex/Doclet is both a multi-format Javadoc doclet and a free edition of DocFlex/Javadoc. If you need to customize your Javadoc without writing a full-blown doclet from scratch, DocFlex/Javadoc may be the only tool able to help you! Find out more at [www.docflex.com](http://www.docflex.com)
8 Considered Alternatives

8.1 EventAdmin integration
This section has moved into the Considered Alternatives chapter as it's postponed to a later release.

Send EventAdmin event using CDI events

EventAdmin and CDI events are conceptually similar. The CDI programming model is much more user-friendly however. EventAdmin events can both be produced and observed using CDI annotations. Because not every event should be published to EventAdmin the developer has to use the @EventAdmin annotation. The value of this annotation should also contain the name of the Topic, or alternatively you can define your own qualifier that extends @EventAdmin to define the Topic.

The following is an example of using the @EventAdmin qualifier.

```java
@Inject @EventAdmin("MyTopic") Event<MyEvent> event;
public void send() {
    event.fire(new MyEvent("example"));
}
```

The following is an example of extending the EventAdmin qualifier.

```java
@EventAdmin
public @interface Demo {
    String value();
}
@Inject @Demo Event<MyEvent> event;
```
public void send() {
    event.fire(new MyEvent("example"));
}

The CDI-OSGi bridge observes @EventAdmin events and republish them as EventAdmin events.

Listener to EventAdmin events using CDI observers

EventAdmin events can be observed using the CDI @Observes annotation. Similar to publishing EventAdmin events we need the @EventAdmin qualifier to specify the Topic name.

public class EventExample {
    public void process(@Observes @EventAdmin("MyTopic") MyEvent event) {}  
}

Alternatively, similar with publishing events, a qualifier can be used to define the topic name.

@EventAdmin
public @interface Demo {
    String value();
}

class EventExample {

public void process(@Observes @Demo MyEvent event) {}
Abstract

The Enterprise JavaBeans architecture (EJB), part of JEE technology, is a managed server-side component architecture for the construction of enterprise applications[4]. While OSGi moves towards enterprise, it has long been interest in integrating EJB into OSGi. This RFP describes the requirements for integrating EJB into the OSGi framework with the intention that the current EJB developers should be able to reuse their current skills and assets to develop and deploy EJ Bs that run in and exploit the OSGi framework.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

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

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

<table>
<thead>
<tr>
<th>0 Document Information</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 License</td>
<td>2</td>
</tr>
<tr>
<td>0.2 Trademarks</td>
<td>3</td>
</tr>
<tr>
<td>0.3 Feedback</td>
<td>3</td>
</tr>
<tr>
<td>0.4 Table of Contents</td>
<td>3</td>
</tr>
<tr>
<td>0.5 Terminology and Document Conventions</td>
<td>4</td>
</tr>
<tr>
<td>0.6 Revision History</td>
<td>4</td>
</tr>
</tbody>
</table>

1 Introduction

2 Application Domain

2.1 Terminology + Abbreviations
2.2 EJB
2.3 EJB Example
2.4 Modular EJB in Apache Aries
2.5 Modular EJB in Glassfish

3 Problem Description

4 Requirements

4.1 Requirements Derived from RFP 98 [6]
4.2 Requirements derived from RFP 0146[7]
5 Technical Solution

5.1 Architecture overview

5.1.1 EJB Bundle

5.1.2 Fragment bundles

5.2 EJB bundle life cycle

5.2.1 Installing an EJB bundle

5.2.2 Starting an EJB bundle

5.2.3 Stopping an EJB bundle

5.2.4 Uninstalling an EJB bundle

5.3 Registering services in Service registry

5.4 OSGi EJB Container

5.4.1 Resource lookup

5.4.2 Resource injection and annotations

5.4.3 EJB application class loader

5.5 Component model interoperability

6 Data Transfer Objects

7 Considered Alternatives

8 Security Considerations

9 Document Support

9.1 References

9.2 Author's Address

9.3 Acronyms and Abbreviations

9.4 End of Document

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

0.6 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>10/31/2012</td>
<td>Emily Jiang, IBM, <a href="mailto:emijiang@uk.ibm.com">emijiang@uk.ibm.com</a></td>
</tr>
<tr>
<td>0.1</td>
<td>03/10/12</td>
<td>Emily Jiang, rework based on the feedback from Austin f2f on 30th Jan 2013</td>
</tr>
<tr>
<td>0.2</td>
<td>03/13/12</td>
<td>Emily Jiang, updated as per discussion on EEG call (13/03/2012)</td>
</tr>
</tbody>
</table>
1 Introduction

The Enterprise JavaBeans architecture (EJB) is an architecture for the development and deployment of component-based business applications[3]. EJB is a part of Java EE that already is well used for developing enterprise applications. However, EJB has limited support for modularity and life-cycle dynamics. With the increase in popularity of OSGi for developing and deploying enterprise applications, the need arises for combining these two technologies. There has long been interest in EJB in the context of OSGi and some application servers such as WebSphere Application Server, GlassFish, OW2 Easybeans and OW2 JOnAS have their own support for EJB in OSGi applications. An OSGi standard way should be defined in order for applications to be portable. This RFP defines the requirements for such a standard.

2 Application Domain

Many enterprise applications have been written using the EJB component model. These applications are often large and require modification from time-to-time, as bugs are fixed or the needs of the business change. Lack of strong modularity in Java EE makes maintenance of large EJB applications difficult. Developers need to be able to build EJB applications in a more modular way. In doing so, they want to re-use their existing EJB assets and skills and only have to learn the new modularity features. Deployers need to be able to deploy these modular EJB applications in ways that leverage the modularity, for example, to manage versioned fixes to an application.

2.1 Terminology + Abbreviations

EJB – Enterprise JavaBean.

EJB Bundle – An OSGi bundle that contains EJBs.

2.2 EJB

The EJB specification is part of Java EE specification. The EJB Specification was originally developed in 1997 and later EJB1.1, EJB2.0 (JSR19), EJB2.1 (JSR153), EJB3.0 (JSR220) and most recent EJB 3.1 (JSR 318) further improved the design. EJBs are deployed and executed in an EJB container, which can be provided by a Java EE Application server.

2.3 EJB Example

There are three types of Enterprise beans: Session Beans, Message Driven Beans and Entity Beans (deprecated). Below is an example of stateless session bean, using EJB 3 annotations.
/**
 * Local stateless session Order EJB
 */
@Local
@Stateless
public class OrderEJB implements Order {
...
@Override
public String process() {
  ...
  return "Order Processed";
}
}

2.4 Modular EJB in Apache Aries

The Apache Aries project ([http://aries.apache.org](http://aries.apache.org)) has created an integration between EJB and OSGi, which introduced the concept of an EJB bundle. An EJB bundle is a bundle that contains EJBs. The EJB bundle is identified by the presence of ‘Export-EJB’ header in the bundle manifest.

The value of this header specifies the stateless or singleton session beans to be registered in the OSGi service registry. The meaning of the header is as follows:

NONE: process all EJBs but register none of them in the service registry.

A single space: register all EJBs in the service registry.

A comma separated list of class names of the EJBs to be registered in the service registry.

Below is an example of MANIFEST.MF for the Order Bean. The OrderEJB will be registered in the service registry under the interface of `com.acme.order.api.Order`.

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Acme order processing service
Bundle-SymbolicName: com.acme.order.service
Bundle-Version: 1.0.0
Export-EJB: OrderEJB
Import-Package:
  com.acme.order.api;version="[1.0.0,1.1.0)",
  javax.ejb;version="3.1"

An EJB is registered in OSGi service registry with the following properties:

ejb.name – The name of the EJB

ejb.type – The type of the EJB, which is either Stateless or Singleton.

Service.exported.interfaces: The remote interfaces of the exported EJB. This enables the EJB to be accessed outside the enterprise application using the OSGi Remote Services specification.

The EJB lookup mechanisms in OSGi continues to support the traditional EJB lookup such as @EJB and JNDI (java:comp/env). With the EJB presence in OSGi service registry, the service look up via (osgi:service) is supported as well.
Traditional EJB look up:

```java
/**
 * Order EJB to be injected
 */
@EJB
private Order orderService = null;

or

orderService =
(Order) new InitialContext().lookup("java:comp/env/com.acme.order.View/orderService");

EJB look up via service registry

orderService =
(Order) new InitialContext().lookup("osgi:service/com.acme.order.api.Order");
```

### 2.5 Modular EJB in Glassfish

Glassfish v3 supports EJB bundles. Similar to Apache Aries, an EJB bundle is identified by the presence of the Manifest header of ‘Export-EJB’. The value of this header specifies the stateless or singleton session beans to be registered in the OSGi service registry. The implementation does not support remote interfaces. The meaning of the header is as follows.

- **ALL**: all local Stateless and Singleton EJBs exported with local business interfaces.
- **NONE**: no EJBs are exported.
- **A list of EJB names**: the EJBs listed here are exported.

GlassFish supports the injection of OSGi services into an EJB by extending the Context and Dependency Injection (CDI) framework.

```java
@Stateless

public class GF_EJB {
    @Inject @OSGiService(Dynamic=true)
    Bar bar;

    ...
}
```
3 Problem Description

EJB is a standard component architecture for building object oriented and distributed business applications in Java. The EJB container provides system-level services such as persistence, transaction and security while the EJB developers can concentrate on the business logic.

The HTTP Service and Web Application Specifications define how web applications can exploit OSGi, which cover some use cases that integrate Java EE technology. Often, such applications use EJB as the component model for business logic and so the use of EJBs in OSGi helps complete the set of Java EE technologies that enterprise OSGi users are looking for.

4 Requirements

EJB001 – The solution MUST make it possible to use the EJB annotations (or XML equivalent) in an OSGi bundle to expose and consume EJBs.

EJB002 – The solution MUST make it possible to publish EJBs in the OSGi Service Registry.

EJB003 – The solution MUST make it possible to consume OSGi services in an EJB

EJB004 – The solution MUST make it possible to select OSGi services used in EJBs based on OSGi filters.

EJB005 – The solution MUST make it possible to support JNDI look up.

EJB006 – The solution MUST make it possible to write a portable EJB jar that runs both in Java EE as well as in OSGi.

EJB007 – The solution MUST consider the thread-safety issues that can arise when migrating EJB beans from Java EE to OSGi.

EJB008 – The solution MUST consider the issues that can arise in relation to the dynamic bundle lifecycle in OSGi.

EJB009 – The solution MUST make it possible to take advantage of the dynamic service capabilities of OSGi.
EJB010 – The solution MAY introduce additional annotations.

EJB011 – The solution MUST define the behavior in case of incorrect EJB metadata.

EJB013 – The solution MUST define an opt-in mechanism. Bundles not opting in MUST not be considered by the EJB-OSGi integration layer.

EJB014 – The solution MUST support the simplified APIs of EJB3.x integration in OSGi but MUST NOT prevent a compliant implementation from also supporting earlier EJB APIs.

EJB015 – The solution SHOULD provide a mechanism to specify additional OSGi service registration properties for EJB beans.

EJB016 – All the inter-bundle interactions with EJBs - MUST go through the OSGi Service Registry.

EJB017 – The solution MUST consider the issues that can arise with OSGi service dynamism when these services are injected into a EJB bean.

EJB018 – The solution MUST provide a mechanism to consume multiple matching services/beans in an EJB bean.

EJB019 - The solution MUST support the transaction, security, persistence and lifecycle semantics associated with EJBs.

EJB020 - The solution MUST support the standard EJB injection models.

EJB021 - The solution MUST support service registration of stateless and singleton EJBs and MAY support service registration of other EJB types.

EJB022 – The solution MUST support local and remote EJBs.

4.1 Requirements Derived from RFP 98 [6].

EJB023 – The solution SHOULD enable the standard EJB artifacts to remain installed when a supporting EJB container is dynamically replaced.

EJB024 – The solution MUST NOT impede the ability of an OSGi-compliant EJB container implementation to also be compliant with the Java EE EJB specification.

EJB025 – The solution MAY define the additional aspects that are required for EJB to be properly integrated in an OSGi framework but MUST NOT make any syntactic changes to the Java interfaces defined by the EJB specifications.

EJB026 – The solution MUST NOT require an OSGi Execution Environment greater than that which satisfies only the signatures of EJB specification.
4.2 Requirements derived from RFP 0146[7].

EJB027 – The solution MUST support CDI Integration.

EJB028 – The solution MUST Support the standard CDI injection.

5 Technical Solution

5.1 Architecture overview

Bundle is the deployment unit in OSGi. This RFC takes a design approach where an EJB Jar is deployed as an OSGi bundle in OSGi framework. This bundle is referred to as EJB bundle throughout the specification.

An EJB bundle is an OSGi bundle that contains EJBs. This specification describes the design requirements for an OSGi EJB container that supports EJB applications written to the EJB specification. The EJB container itself is deployed as one or more OSGi bundles.

The design uses the OSGi extender pattern[8], where the EJB container acts as an extender that is responsible for observing the life cycle of EJB bundles. The EJB extender provides the capability of osgi.ejb.extender namespace with the following manifest header.

| Provide-Capability: osgi.extender; osgi.extender="osgi.ejb" |

When an EJB bundle is started, the EJB container processes the configuration files of the application and instantiates and manages the lifecycle of EJBs in the EJB bundle. The EJB Extender manages the EJB bundles.

5.1.1 EJB Bundle

An EJB bundle is defined as a normal bundle that contains EJBs. It is identified by declaring a requirement on the EJB extender. This requirement activates the extender to inspect the EJB bundle's export.ejb directive. The directive of 'export.ejb' specifies which EJBs will be registered in the OSGi Service Registry.

It is identified by the opt-in header of Export EJB. The following table lists the meaning of the Export EJB header.

<table>
<thead>
<tr>
<th>Export-EJB export.ejb: (one space)*</th>
<th>Process all EJBs and register them in the service registry</th>
</tr>
</thead>
<tbody>
<tr>
<td>export.ejb Export-EJB: &lt;comma separated ejb names&gt; (e.g. orderEJB, colorEJB)</td>
<td>Process all EJBs but only register the listed EJBs in the service registry</td>
</tr>
<tr>
<td>:NONE Export-EJB</td>
<td>Process all EJBs but do not register any EJBs in the service registry</td>
</tr>
<tr>
<td>Absence</td>
<td>Process all EJBs but do not register any EJBs in the service registry. Do not process any EJBs in the bundle</td>
</tr>
</tbody>
</table>
Require-Capability: osgi.extender; filter="(osgi.extender=osgi.ejb)"; export.ejb="*"

An EJB Bundle with the header of Export-EJB: registers services with the following service properties. The service properties are detailed below.

<table>
<thead>
<tr>
<th>Key</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ejb.name</td>
<td>String</td>
<td>The name of the EJB</td>
</tr>
<tr>
<td>ejb.type</td>
<td>String</td>
<td>The type of the EJB (e.g. Stateless, Stateful, Singleton)</td>
</tr>
<tr>
<td>service.exported.interfaces</td>
<td>String+</td>
<td>The remote interfaces the EJB implements</td>
</tr>
<tr>
<td>service.exported.configs</td>
<td>StringBoolean</td>
<td>osgi.ejb means the EJB container does the remoting and its rmi-riop. Specify whether the EJB container does the remoting</td>
</tr>
</tbody>
</table>

An EJB bundle is defined as follows:

- An EJB is a valid OSGi bundle and as such must fully describe its dependencies.
- An EJB bundle follows the OSGi bundle life cycle.
- An EJB bundle is differentiated from a normal bundle through the presence of Export-EJB manifest header requirement on the EJB extender. The header export.ejb directive specifies the EJBs to be registered in the OSGi Service Registry. The EJB bundle can contain session beans and MDBs. The processing of entity EJB is not required.
- An EJB bundle supports EJB3.x beans and this specification does not mandate the support of the earlier EJB versions, e.g. EJB2.1.

5.1.2 Fragment bundles

An OSGi fragment is a Jar file with specific manifest headers that enable it to attach to a specified host bundle or specified host bundles in order to function. Fragments are treated as part of the host bundles. Relevant definitions of the fragment are merged with the host bundles definitions before the host is resolved, as long as the information does not conflict. Fragments extend bundles with resources, classes, and permitted headers enabling you to customize your bundles. This specification limits what effects a fragment bundle can have on a EJB bundle in the following way:

- A fragment can contribute additional EJBs to a host bundle.
- A fragment cannot replace the ejb-jar.xml for a host bundle. If there are conflicts, the host configuration wins.
- A fragment can contribute classes to the bundle class space.

Copyright © OSGi Alliance 2013 All Rights Reserved
5.2 EJB bundle life cycle

5.2.1 Installing an EJB bundle

An EJB bundle is a valid OSGi bundle whose manifest specifies the requirement of the EJB extender. An EJB bundle can be installed into the framework using the standard BundleContext.installBundle API variants. Once installed, an EJB bundle's life cycle is managed just like any other bundle in the framework.

5.2.2 Starting an EJB bundle

An EJB application is started by starting its EJB bundles. The EJB Extender listens for the bundle starting and lazy activation life cycle events to initiate this process. A bundle that has a lazy activation policy should not be transitioned to the STARTED state by the EJB Extender until a request is made that requires a class to be loaded.

The EJB Extender recognizes an EJB bundle by looking for the presence of Export-EJB manifest header. The extender may not recognizes a bundle as an EJB bundle unless the Export-EJB header is present in its manifest.

After recognizing an EJB bundle, the extender initiates the process of deploying the application into an EJB container. It must generate a DEPLOYING event as described in section x. The container must validate the export.ejb directive to ensure the EJBs can be located in the bundle. Otherwise, a FAILED event must be emitted. After successfully deploying the application, a DEPLOYED event must be generated to indicate that the application is now in service.

5.2.3 Stopping an EJB bundle

An EJB application is stopped by simply stopping the corresponding EJB bundle. In response to a bundle STOPPING event, the extender must initiate the process of undeploying the application from the EJB container and clean up any resources.

- An UNDEPLOYING event is emitted to state that the application will be removed.
- EJB container must clean up any application specific resources.
- Finally, an UNDEPLOYED event is emitted.

Once the bundle is stopped, the OSGi framework will automatically unregistered any services registered by the EJB application.

5.2.4 Uninstalling an EJB bundle

An EJB application can be uninstalled by uninstalling the corresponding EJB bundle. The application bundle will be uninstalled from the OSGi framework, and will be completely removed when the framework is refreshed.

5.3 Integration with EventAdmin service
If the EventAdmin service is registered then the EJB extender bundle must emit the following events:

- `org/osgi/service/ejb/DEPLOYING` – the EJB extender has spotted an EJB bundle and started the process of deploying the EJB application.
- `org/osgi/service/ejb/DEPLOYED` – the EJB extender has finished deploying the EJB application, and the application is now running (in service).
- `org/osgi/service/ejb/UNDEPLOYING` – the EJB extender started removing the EJB application in response to the bundle being stopped.
- `org/osgi/service/ejb/UNDEPLOYED` – the EJB extender has removed the EJB application. The application is no longer in service.
- `org/osgi/service/ejb/FAILED` - the EJB extender has failed to deploy the EJB application, this will be emitted after the DEPLOYING event has fired.

For each event the following properties must be published:

- “bundle.symbolicName” (String) the symbolic name of the EJB bundle.
- “bundle.id” (Long) the id of the EJB bundle.
- “bundle” (Bundle) the Bundle object of the EJB bundle.
- “bundle.version” (Version) the version of the EJB bundle.
- “timestamp” (Long) the time when the event occurred
- “extender.bundle” (Bundle) the Bundle object of the EJB Extender bundle
- “extender.bundle.id” (Long) the id of the EJB Extender bundle
- “extender.bundle.symbolicName” (String) the symbolic name of the EJB Extender bundle.
- “extender.bundle.version” (Version) the version of the EJB Extender bundle.

In addition the FAILED event must also have the following property:
- “exception” (Throwable) an exception detailing the problem.

### 5.3 Registering services in Service registry

When specified in the EJB bundle's manifest header of Export-EJB with the presence of the directive of export.ejb, the EJBs will be registered in the service registry. The entity EJBs and MDBs are not registered in the service registry. If they are listed in the Export-EJB header export.ejb directive, they will be ignored.

There are three types of session beans: Stateful, Stateless and Singleton beans.

For registering stateful and singleton beans, an instance of the beans will be registered in the service registry with the service properties specified in 5.1.1.

When registering the Stateful session beans, the current framework cannot support of creating multiple service instances per consuming bundle, which leads to the existence of RFC Service Scope [9]. The RFC Service Scope introduces a third scope: prototype, besides the existing two scopes, **singleton and bundle**. Prototype scope is used by specifying the service.scope property with 'prototype' when registering Stateful session beans.

A stateful session bean needs to implement the new type of PrototypeServiceFactory and then registers a PrototypeServiceFactory object as the service object. This service will have the prototype scope.

In the consuming bundle, it can use the following new method on the BundleContext.

```xml
<S> ServiceObjects<S> getServiceObjects(ServiceReference<S> reference)
```
The method of getService() will obtain a new service object for the prototype scope services.

## 5.4 OSGi EJB Container

The specification defines an OSGi EJB container implementation as one or more OSGi bundles that collectively implement EJB 3.x specifications. The following section describes requirement for an OSGi EJB container. The EJBs can be declared either in ejb-jar.xml under META-INF/WEB-INF or via annotations. A consuming client can be either EJB client or Web client and it can use either annotation (@EJB) or ejb-ref in the ejb-jar.xml to obtain an EJB instance.

The specification requires that an OSGi EJB container provide the integration with

- Transaction
- JPA
- Security
- JNDI (look up via java:global, java:app, java:module)

### 5.4.1 Resource lookup

For an EJB bundle, there might be a number of EJBs defined. The EJBs can be defined either via annotations or XML in the EJB deployment descriptor, which is called META-INF/ejb-jar.xml or WEB-INF/ejb-jar.xml in Java EE. The EJBs defined in the host and fragments should be searched through.

### 5.4.2 Resource injection and annotations

The EJB application deployment descriptor may specify the metadata-complete attribute on the ejb-jar element of the ejb-jar.xml. If the attribute value is true, no annotation scan will be performed. Any EJB annotations will be ignored. If the attribute value is false, the container should examine the classes for annotations. This specification must support the EJB 3.x annotations. If JPA, CDI, Security are used in the EJB bundle, the relevant annotations such as @PersistenceContext, @PersistenceUnit for JPA, must be supported.

### 5.4.3 EJB application class loader

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

### 5.5 Component model interoperability

An EJB bundle may need to inter-operate with other component models such as Blueprint services (RFC124) and Declarative Services. Since EJB container owns the life cycle of an EJB bundle, this eliminate the possibility of an EJB bundle to be simultaneously be a component of another component model.
A typical interaction between the different component models is that an EJB bundle depends on a Blueprint or DS service. Interactions between them are managed by the OSGi service registry. An EJB bundle can perform service lookup via BundleContext or via jndi lookup such as osgi:services or @Resource annotation.

6 Data Transfer Objects

RFC 185 defines Data Transfer Objects as a generic means for management solutions to interact with runtime entities in an OSGi Framework. DTOs provides a common, easily serializable representation of the technology.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a DTO? The expectation is that in most cases it would.

The DTOs for the design in this RFC should be described here and if there are no DTOs being defined 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.

7 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 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.
9 Document Support

9.1 References
[6]. RFP 98 OSGi Platform and Java EE Integration.
[7]. RFP 0146 CDI Integration
[9]. RFC 195 Service Scopes

9.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Emily Jiang</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:emijiang@uk.ibm.com">emijiang@uk.ibm.com</a></td>
</tr>
</tbody>
</table>

9.3 Acronyms and Abbreviations

9.4 End of Document
Device Abstraction Layer

Final

84 Pages

Abstract

Defines a new device abstraction API in OSGi platform. It provides a simple access to the devices and their functionality.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0 Document Information.....................................................................................................................2
  0.1 License..................................................................................................................................2
  0.2 Trademarks............................................................................................................................3
  0.3 Feedback...............................................................................................................................3
  0.4 Table of Contents..................................................................................................................3
  0.5 Terminology and Document Conventions...........................................................................4
  0.6 Revision History.....................................................................................................................4

1 Introduction..................................................................................................................................7

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

3 Problem Description..................................................................................................................9

4 Requirements...........................................................................................................................11

5 Technical Solution.....................................................................................................................11
  5.1 Introduction............................................................................................................................11
    5.1.1 Entities..........................................................................................................................12
  5.2 Device Access Category........................................................................................................12
  5.3 Device Service.......................................................................................................................12
    5.3.1 Reference Device Services..........................................................................................16
    5.3.2 Device Service Registration.........................................................................................17
    5.3.3 Device Service Unregistration.....................................................................................17
  5.4 Device Status Transitions.....................................................................................................17
    5.4.1 Transitions to STATUS_REMOVED.............................................................................19
5.4.2 Transitions to and from STATUS_OFFLINE..........................19
5.4.3 Transitions to and from STATUS_ONLINE..........................20
5.4.4 Transitions to and from STATUS_PROCESSING....................21
5.4.5 Transitions to and from STATUS_NOT_INITIALIZED..............22
5.4.6 Transitions to and from STATUS_NOT_CONFIGURED.............23

5.5 Functions.............................................................................24
  5.5.1 Function Interface..........................................................26
  5.5.2 Function Operations.......................................................27
  5.5.3 Function Properties........................................................27
  5.5.4 Function Property Event.................................................28

6 Data Transfer Objects................................................................29

7 Javadoc....................................................................................29

8 Considered Alternatives..........................................................81
  8.1 Use Configuration Admin to update the Device service properties..81
  8.2 DeviceAdmin interface availability.......................................81
  8.3 Access helper methods removal of FunctionalDevice..............82

9 Security Considerations...........................................................82
  9.1 Device Permission.............................................................82
  9.2 Required Permissions.........................................................83

10 Document Support......................................................................83
  10.1 References...........................................................................83
  10.2 Author’s Address...............................................................83
  10.3 Acronyms and Abbreviations..............................................84
  10.4 End of Document...............................................................84

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

Source code is shown in this typeface.

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>Jan 22 2013</td>
<td>Initial draft version. Evgeni Grigorov, ProSyst Software, <a href="mailto:e.grigorov@prosyst.com">e.grigorov@prosyst.com</a></td>
</tr>
<tr>
<td>2nd draft</td>
<td>Feb 13 2013</td>
<td>Updated Considered Alternatives and Security Considerations after F2F meeting in Austin, TX. Provide more details about device management. Evgeni Grigorov, ProSyst Software, <a href="mailto:e.grigorov@prosyst.com">e.grigorov@prosyst.com</a></td>
</tr>
<tr>
<td>Revision</td>
<td>Date</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 3<sup>rd</sup> draft | Mar 08 2013 | Remove DeviceAdmin service.  
Describe DeviceFunction and FunctionalDevice interfaces.  
Evgeni Grigorov, ProSyst Software, e.grigorov@prosyst.com |
| 4<sup>th</sup> draft | Apr 08 2013 | Rename the package and some constants.  
Merge the AbstractDevice and FunctionalDevice to FunctionalDevice.  
Add Functional Device Permission.  
Add Device Function Event.  
Minor fixes: renamed Device Access category, fixed unit representation and some clarifications.  
Add a suggestion about Device Functions to be discussed on F2F in Cologne.  
Evgeni Grigorov, ProSyst Software, e.grigorov@prosyst.com |
| 5<sup>th</sup> draft | Jun 12 2013 | Add a basic set of Device Functions.  
Include the device status transitions.  
Update the illustrations.  
Add a status detail mapping.  
Add some snippets.  
Remove the device helper methods for an access to parent, children and reference devices.  
Add a Functional Device and Device Function descriptions.  
Add error codes to DeviceFunctionException.  
Update the javadoc.  
Evgeni Grigorov, ProSyst Software, e.grigorov@prosyst.com |
| 6<sup>th</sup> draft | Jul 02 2013 | Describe the status transitions in detail.  
FunctionalDeviceException CODE_UNKNOWN fixed to CODE_UNKNOWN.  
Functional Group is introduced.  
Functional Device, Functional Group and Device Function are in the service registry.  
New service properties are introduced.  
Parent-child relation is removed.  
Add more details to the descriptions.  
Evgeni Grigorov, ProSyst Software, e.grigorov@prosyst.com |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>7th draft</td>
<td>Sept 09 2013</td>
<td>Basic device function set is updated. Rename FunctionalDevice to Device. Rename FunctionalDeviceException to DeviceException. Rename FunctionalDevicePermission to DevicePermission. Relax the relation between the device and device function. DeviceException extends IOException. Functional group is removed. Renamed device function metadata properties. Evgeni Grigorov, ProSyst Software, <a href="mailto:e.grigorov@prosyst.com">e.grigorov@prosyst.com</a></td>
</tr>
<tr>
<td>8th draft</td>
<td>Jan 16 2014</td>
<td>Service property names are renamed form PROPERTY_&lt;name&gt; to SERVICE_&lt;name&gt;. Status disabled is removed, because it's applicable to small set of devices like peripherals. Remove the public methods to update the device properties. They should be initially configured. Updated permissions, because of updated device management operations. Overview diagram is added. Diagram with all device statuses is added. The package is renamed. Common device function data structure is introduced. Property and operation metadata structures are introduced. Device function type is added. There is a new interface with base set of device function types. There is a new interface with SI unit symbols. Evgeni Grigorov, ProSyst Software, <a href="mailto:e.grigorov@prosyst.com">e.grigorov@prosyst.com</a></td>
</tr>
<tr>
<td>9th draft</td>
<td>Jan 30 2014</td>
<td>Device.setName is removed. The device properties configuration is a vendor specific. Minor javadoc fixes and name improvements after the initial reference implementation. Device Function must be registered under only one interface. WakeUp Device Function is introduced to cover battery-operated devices. Evgeni Grigorov, ProSyst Software, <a href="mailto:e.grigorov@prosyst.com">e.grigorov@prosyst.com</a></td>
</tr>
</tbody>
</table>
1 Introduction

OSGi is gaining popularity as enabling technology for building embedded system in residential and M2M markets. In these contexts it is often necessary to communicate with IP and non-IP devices by using various protocols such as ZigBee, Z-Wave, KNX, UPnP etc. In order to provide a convenient programming model suitable for the realization of end-to-end services it is very useful to define and apply an abstraction layer which unifies the work with devices supporting different protocols.

This RFC defines a new device abstraction API in OSGi.

2 Application Domain

Currently there are several standardization bodies such as OSGi Alliance, HGI, BBF, ETSI M2M which deal with the deployment of services in an infrastructure based on the usage of a Residential Gateway running OSGi as Execution Platform. The picture on Illustration 1 shows a reference architecture which is valid in the majority of cases under consideration.
In this architecture the application logic is distributed between:

- Applications running on the residential gateways
- Applications running on the devices providing UI (e.g. tablets, mobile phones, desktops).

In order to realize services which access other IP and non-IP devices connected to the residential gateway, those applications must be able to read information from the devices and perform operations on them through software APIs. Such an access is essential for services in the area of smart metering, entertainment, home automation, assisted living and security.

The existing OSGi specifications which address related topics are:

- Device Access Specification – focuses on the dynamic discovery of the proper driver when a new device is attached/connected to the residential gateway. The device access is limited to attend the driver installation needs.
- UPnP™ Device Service Specification – defines among the other OSGi API for work with UPnP devices accessible from the residential gateway. API is specified in the scope of UPnP Device Access category.
3 Problem Description

Normally the residential gateways operate in heterogeneous environment including devices that support different protocols. It's not trivial to provide interoperability of the applications and the devices under such circumstances. The existing OSGi Device Access Specification solves the driver installation problems but currently there is no complete API that can be used for accessing the device data and for invoking actions on the devices.

Illustration 2 shows one possible approach for working with heterogeneous devices in an OSGi environment:

In this case each application which accesses devices of a given type must use API specific for this type. One obvious disadvantage of this model is that when a new device protocol is added the applications must be modified in order to support this protocol.

Much better is the approach from Illustration 3 which is defined by this RFC.
In this case an additional device abstraction layer is introduced which unifies the work with the devices provided by the different underlying protocols. Thus the following advantages are achieved:

- The application programmers can work with devices provided by different protocols exactly in the same way and by applying the same program interface. The protocol adapters and device abstraction API hide the complexity/differences of the device protocols.
- The applications can work without modification when new hardware controllers and protocol adapters are dynamically added.
- When remote access to the devices connected to the gateway is necessary (e.g. in m2m and management scenarios) it’s much easier to provide mapping to one API then to a set of protocol dependent APIs.
- It is much easier to build UI for remote browsers or for apps running on mobile devices if just one mapping to one unified device abstraction API is necessary.
4 Requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>The solution MUST define API for controlling devices which is applicable for all relevant device protocols.</td>
</tr>
<tr>
<td>2.</td>
<td>The solution MUST define API for controlling devices which is independent from the device protocols.</td>
</tr>
<tr>
<td>3.</td>
<td>The solution MUST include device access control based on user and application permissions compliant with the OSGi security model.</td>
</tr>
<tr>
<td>4.</td>
<td>The solution MUST take advantage of the security features available in the device protocols.</td>
</tr>
<tr>
<td>5.</td>
<td>The solution MUST include a device protocol independent notification mechanism realized according to the OSGi event mechanisms.</td>
</tr>
<tr>
<td>6.</td>
<td>The solution SHOULD be mappable to other relevant standards such as HGI, ETSI M2M and BBF handling the remote access to device networks.</td>
</tr>
<tr>
<td>7.</td>
<td>The solution MUST provide configurable device data and metadata model.</td>
</tr>
<tr>
<td>8.</td>
<td>The solution MUST be applicable to the changeable device behavior. Sleeping/power saving devices can go and stay offline for a long time, but should be available in the defined API.</td>
</tr>
<tr>
<td>9.</td>
<td>The solution MUST provide an extension mechanism to support devices provided by new protocols.</td>
</tr>
<tr>
<td>10.</td>
<td>The solution MAY provide means to access the protocol specific device object.</td>
</tr>
<tr>
<td>11.</td>
<td>The solution MUST register device or/and device related instance to the OSGi service registry.</td>
</tr>
</tbody>
</table>

5 Technical Solution

5.1 Introduction

Remote device control provides opportunity to save energy, to provide better security, to save your time during daily tasks and many more. The devices can play different roles in their networks as events reporters, controllers etc. That dynamic behavior is well mappable to the dynamic OSGi service registry. There is a registration of Device service. It realizes basic set of management operations and provides rich set of properties. The applications are allowed to track the device status, to read descriptive information and to follow the device relations. A set of functions can belong to the device. They represents the device operations and related properties in an atomic way. The device functions can be found in the OSGi service registry. The applications are
allowed to get directly the required functions if they don't need information about the device. For example, light
device is registered as a Device service and there is a Function service to turn on and turn off the light.

5.1.1 Entities

- Device – represents the device in the OSGi service registry. It’s described with a set of service properties
  and provides basic management operations.
- Function – atomic functional entity. The device can support a few functions like switch and sensor. The
  function provides a set of properties and operations.
- FunctionEvent – asynchronous event. It’s sent through EventAdmin service and notifies for Function
  property change.
- FunctionData – data structure carries Function property value with additional metadata.
- PropertyMetadata and OperationMetadata – contains metadata about the Function properties and
  operations.

5.2 Device Access Category

The device access category is called “DAL”. The category name is defined as a value of
Device.DEVICE_CATEGORY constant. It can be used as a part of
org.osgi.service.device.Constants.DEVICE_CATEGORY service property key value. The category
impose this specification rules.

5.3 Device Service

Device interface is dedicated for a common access to the devices provided by different protocols. It can be
mapped one to one with the physical device, but can be mapped only with a given functional part of the device. In
this scenario, the physical device can be realized with a set of Device services and different relations between
them. Device service can represent pure software unit. For example, it can simulate the real device work. There
are basic management operations for remove, property access and property update. New protocol devices can be
supported with a registration of new Device services.
If the underlying protocol and the implementation allow, the Device services must be registered again after the OSGi framework reboot. The service properties must be restored, the supported functions must be provided and Device relations must be visible to the applications.

The OSGi service registry has the advantage of being easily accessible. The services can be filtered and accessed with their properties. The device service has a rich set of such properties as it is on Illustration 5:

- **Device.SERVICE_UID** – Specifies the device unique identifier. It's a mandatory property. The value type is `java.lang.String`. To simplify the unique identifier generation, the property value must follow the rule:
  
  \[
  \text{UID ::= driver-name \cdot device-id}
  \]

  - `UID` – device unique identifier
  - `driver-name` – the value of the `Device.SERVICE_DRIVER` service property
  - `device-id` – device unique identifier in the scope of the driver

- **Device.SERVICE_REFERENCE_UIDS** – Specifies the reference device unique identifiers. It's an optional property. The value type is `java.lang.String[]`. It can be used to represent different relationships between the devices. For example, The ZigBee controller can have a reference to the USB dongle.

- **Device.SERVICE_DRIVER** – Specifies the device driver name. For example, ZigBee, Z-Wave, Bluetooth etc. It's a mandatory property. The value type is `java.lang.String`.

- **Device.SERVICE_NAME** – Specifies the device name. It's an optional property. The value type is `java.lang.String`.

- **Device.SERVICE_STATUS** – Specifies the current device status. It's a mandatory property. The value type `java.lang.Integer`. The possible values are:

  - **Device.STATUS_REMOVED** – Indicates that the device is removed from the network. That status must be set as the last device status and after that the device service can be unregistered from the service registry. The status is available for stale device services too. All transitions to and from this status are described in Transitions to STATUS_REMOVED section.

  - **Device.STATUS_OFFLINE** – Indicates that the device is currently not available for operations. The end device is still installed in the network and can become online later. The controller is unplugged or there is no connection. All transitions to and from this status are described in detail in Transitions to and from STATUS_OFFLINE section.

  - **Device.STATUS_ONLINE** – Indicates that the device is currently available for operations. All transitions to and from this status are described in detail in Transitions to and from STATUS_ONLINE section.

  - **Device.STATUS_PROCESSING** – Indicates that the device is currently busy with an operation. All transitions to and from this status are described in detail in Transitions to and from STATUS_PROCESSING section.

  - **Device.STATUS_NOT_INITIALIZED** – Indicates that the device is currently not initialized. Some protocols don't provide device information right after the device is connected. The device can be initialized later when it's awakened. All transitions to and from this status are described in detail in Transitions to and from STATUS_NOT_INITIALIZED section.

  - **Device.STATUS_NOT_CONFIGURED** – Indicates that the device is currently not configured. The device can require additional actions to become completely connected to the network. All transitions to and from this status are described in detail in Transitions to and from STATUS_NOT_CONFIGURED section.
• **Device.SERVICE_STATUS_DETAIL** – Provides the reason for the current device status. It’s an optional property. The property value cannot be externally set or modified. The value type is java.lang.Integer. There are two value categories. Positive values indicate the reason for the current status like Device.STATUS_DETAIL_CONNECTING. Negative values indicate errors related to the current device status like Device.STATUS_DETAIL_DEVICE_BROKEN. The list with defined status details is:

  - **Device.STATUS_DETAIL_CONNECTING** – The reason for the current device status is that the device is currently connecting to the network. It indicates the reason with a positive value 1. The device status must be STATUS_PROCESSING.

  - **Device.STATUS_DETAIL_INITIALIZING** – The reason for the current device status is that the device is currently in process of initialization. It indicates the reason with a positive value 2. The network controller initializing means that information about the network is currently read. The device status must be STATUS_PROCESSING.

  - **Device.STATUS_DETAIL_REMOVING** – The reason for the current device status is that the device is leaving the network. It indicates the reason with positive value 3. The device status must be STATUS_PROCESSING.

  - **Device.STATUS_DETAIL_CONFIGURATION_NOT_APPLIED** – The reason for the current device status is that the device configuration is not applied. It indicates an error with a negative value -1. The device status must be STATUS_NOT_CONFIGURED.

  - **Device.STATUS_DETAIL_DEVICE_BROKEN** – The reason for the offline device is that the device is broken. It indicates an error with a negative value -2. The device status must be STATUS_OFFLINE.

  - **Device.STATUS_DETAIL_DEVICE_COMMUNICATION_ERROR** – The reason for the current device status is that the device communication is problematic. It indicates an error with a negative value -3. The device status must be STATUS_ONLINE or STATUS_NOT_INITIALIZED.

  - **Device.STATUS_DETAIL_DEVICE_DATA_INSUFFICIENT** – The reason for the uninitialized device is that the device doesn’t provide enough information and cannot be determined. It indicates an error with a negative value -4. The device status must be STATUS_NOT_INITIALIZED.

  - **Device.STATUS_DETAIL_DEVICE_NOT_ACCESSIBLE** – The reason for the offline device is that the device is not accessible and further communication is not possible. It indicates an error with a negative value -5. The device status must be STATUS_OFFLINE.

  - **Device.STATUS_DETAIL_ERROR_APPLYING_CONFIGURATION** – The reason for the current device status is that the device cannot be configured. It indicates an error with a negative value -6. The device status must be STATUS_NOT_CONFIGURED.

  - **Device.STATUS_DETAIL_IN_DUTY_CYCLE** – The reason for the offline device is that the device is in duty cycle. It indicates an error with a negative value -7. The device status must be STATUS_OFFLINE.

Custom status details are allowed, but they must not overlap the specified codes. Table 1 contains the mapping of the status details to the statuses.
<table>
<thead>
<tr>
<th>Status Detail</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_DETAIL_CONNECTING</td>
<td>STATUS_PROCESSING</td>
</tr>
<tr>
<td>STATUS_DETAIL_INITIALIZING</td>
<td>STATUS_PROCESSING</td>
</tr>
<tr>
<td>STATUS_DETAIL_REMOVING</td>
<td>STATUS_PROCESSING</td>
</tr>
<tr>
<td>STATUS_DETAIL_CONFIGURATION_NOT_APPLIED</td>
<td>STATUS_NOT_CONFIGURED</td>
</tr>
<tr>
<td>STATUS_DETAIL_DEVICE_BROKEN</td>
<td>STATUS_OFFLINE</td>
</tr>
<tr>
<td>STATUS_DETAIL_DEVICE_COMMUNICATION_ERROR</td>
<td>STATUS_ONLINE, STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>STATUS_DETAIL_DEVICE_DATA_INSUFFICIENT</td>
<td>STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>STATUS_DETAIL_DEVICE_NOT_ACCESSIBLE</td>
<td>STATUS_OFFLINE</td>
</tr>
<tr>
<td>STATUS_DETAIL_ERROR_APPLYING_CONFIGURATION</td>
<td>STATUS_NOT_CONFIGURED</td>
</tr>
<tr>
<td>STATUS_DETAIL_IN_DUTY_CYCLE</td>
<td>STATUS_OFFLINE</td>
</tr>
</tbody>
</table>

Table 1

- Device.SERVICE_HARDWARE_VENDOR – Specifies the device hardware vendor. It's an optional property. The value type is java.lang.String.
- Device.SERVICE_HARDWARE_VERSION – Specifies the device hardware version. It's an optional property. The value type is java.lang.String.
- Device.SERVICE_FIRMWARE_VENDOR – Specifies the device firmware vendor. It's an optional property. The value type is java.lang.String.
- Device.SERVICE_FIRMWARE_VERSION – Specifies the device firmware version. It's an optional property. The value type is java.lang.String.
- Device.SERVICE_TYPES – Specified the device types. It's an optional property. The value type is java.lang.String[].
- Device.SERVICE_MODEL – Specifies the device model. It's an optional property. The value type is java.lang.String.
- Device.SERVICE_SERIAL_NUMBER – Specifies the device serial number. It's an optional property. The value type is java.lang.String.

The device services are registered in the OSGi service registry with org.osgi.service.dal.Device interface. The next code snippet prints the online devices.

```java
final ServiceReference[] deviceSRefs = context.getServiceReferences(
    Device.class.getName(),
    '( ' + Device.SERVICE_STATUS + ' = ' + Device.STATUS_ONLINE + ' )');
if (null == deviceSRefs) {
    return; // no such services
}
for (int i = 0; i < deviceSRefs.length; i++) {
```
Applications need to have an access to the device properties. For convenience there is a helper method:

- **getServiceProperty(String propName)** – Returns the current value of the specified property. The method will return the same value as `org.osgi.framework.ServiceReference.getProperty(String)` for the service reference of this device.

5.3.1 Reference Device Services

Device service can have a reference to other devices. That link can be used to represent different relationships between devices. For example, the ZigBee dongle can be used as USB Device and ZigBee network controller...
Device. The network controller device can have a reference to the physical USB device as it's depicted on Illustration 6.

The related service property is `Device.SERVICE_REFERENCE_UIDS`.

![Network Controller](Network Controller has reference USB Device)

**Illustration 6**

### 5.3.2 Device Service Registration

The devices are registered as services in the OSGi service registry. The service interface is `org.osgi.service.dal.Device`. There is a registration order. Device services are registered last. Before their registration, there is Function service registration.

### 5.3.3 Device Service Unregistration

OSGi service registry is only about the read-only access for the services. There are no control operations. The service provider is responsible to register, update or unregister the services. That design is not very convenient for the device life cycle. The `Device` interface provides a callback method `remove()`. The method can be optionally implemented by the device provider. `java.lang.UnsupportedOperationException` can be thrown if the method is not supported. When the remove callback is called, an appropriate command will be synchronously send to the device. As a result it can leave the network and device related service will be unregistered. There is an unregistration order. The registration reverse order is used when the services are unregistered. Device services are unregistered first before Function services.

### 5.4 Device Status Transitions

The device status uncover the device availability. It can demonstrate that device is currently not available for operations or that the device requires some additional configuration steps. The status can jump over the different values according to the rules defined in this section. The status transitions are summarized in Table 2, visualized in Illustration 7 and described in detail in the next sections. The entry device status is always `STATUS_PROCESSING`. When the device info is processed, the device can go to another status. The last possible device status is `STATUS_REMOVED`. The status must be set when the device is removed from the network. After that status, the device service will be unregistered.
Illustration 7
### Table 2

<table>
<thead>
<tr>
<th>From \ To Status</th>
<th>PROCESSING</th>
<th>ONLINE</th>
<th>OFFLINE</th>
<th>NOT_INITIALIZED</th>
<th>NOT_CONFIGURED</th>
<th>REMOVED</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESSING</td>
<td>-</td>
<td>Initial device data has been read.</td>
<td>Device is not accessible.</td>
<td>Initial device data is partially read.</td>
<td>Device has a pending configuration.</td>
<td>Device is removed.</td>
</tr>
<tr>
<td>ONLINE</td>
<td>Device data is processing.</td>
<td>-</td>
<td>Device is not accessible.</td>
<td>-</td>
<td>Device has a new pending configuration.</td>
<td>Device is removed.</td>
</tr>
<tr>
<td>OFFLINE</td>
<td>Device data is processing.</td>
<td>Device data has been read.</td>
<td>-</td>
<td>-</td>
<td>Device has a pending configuration.</td>
<td>Device is removed.</td>
</tr>
<tr>
<td>NOT_INITIALIZED</td>
<td>Device data is processing.</td>
<td>-</td>
<td>Device is not accessible.</td>
<td>-</td>
<td>-</td>
<td>Device is removed.</td>
</tr>
<tr>
<td>NOT_CONFIGURED</td>
<td>Device data is processing.</td>
<td>Device pending configuration is satisfied.</td>
<td>Device is not accessible.</td>
<td>-</td>
<td>-</td>
<td>Device is removed.</td>
</tr>
<tr>
<td>REMOVED</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

#### 5.4.1 Transitions to STATUS_REMOVED

The device can go to `Device.STATUS_REMOVED` from any other status. Once reached, the device status cannot be updated any more. The device is removed from the network and the device service is unregistered from the OSGi service registry. If there are stale references to the `Device` service, their status will be set to `STATUS_REMOVED`.

The common way for a given device to be removed is `Device.remove()`. When the method returns, the device status will be `STATUS_REMOVED`. It requires a synchronous execution of the operation.

#### 5.4.2 Transitions to and from STATUS_OFFLINE

The `STATUS_OFFLINE` indicates that the device is currently not available for operations. That status can be set, because of different reasons. The network controller can be unplugged, connection to the device is lost etc. This variety provides an access to that status from any other except `STATUS_REMOVED`. Transitions to and from this status are:

- From `STATUS_OFFLINE` to `STATUS_REMOVED` – device is removed. The status can be set as a result of `Device.remove()` method call.
- From `STATUS_OFFLINE` to `STATUS_PROCESSING` – device data is processing.
- From `STATUS_OFFLINE` to `STATUS_NOT_CONFIGURED` – device has a pending configuration.
- From `STATUS_OFFLINE` to `STATUS_ONLINE` – device data has been read and the device is currently available for operations.
- From `STATUS_OFFLINE` to `STATUS_NOT_INITIALIZED` – That transition is not possible, because the status have to go through `STATUS_PROCESSING`. If the processing is unsuccessful, `STATUS_NOT_INITIALIZED` will be set.
5.4.3 Transitions to and from STATUS_ONLINE

The STATUS_ONLINE indicates that the device is currently available for operations. The online devices are initialized and ready for use. Transitions to and from this status are:

- From STATUS_ONLINE to STATUS_REMOVED – device is removed. The status can be set as a result of Device.remove() method call.
- From STATUS_ONLINE to STATUS_PROCESSING – device data is processing.
- From STATUS_ONLINE to STATUS_NOT_CONFIGURED – device has a pending configuration.
- From STATUS_ONLINE to STATUS_OFFLINE – Online device is not accessible any more.
- From STATUS_ONLINE to STATUS_NOT_INITIALIZED – That transition is not possible. Online devices are initialized.
- To STATUS_ONLINE from STATUS_REMOVED – That transition is not possible. If device is removed, the service will be unregistered from the service registry.
- To STATUS_ONLINE from STATUS_PROCESSING – Initial device data has been read. The device is available for operations.
- To STATUS_ONLINE from STATUS_NOT_CONFIGURED – The device pending configuration is satisfied.
- To STATUS_ONLINE from STATUS_OFFLINE – device is accessible for operations.
- To STATUS_ONLINE from STATUS_NOT_INITIALIZED – That transition is not possible. The device data has to be processed and then the device can become online. Intermediate status STATUS_PROCESSING will be used.

The possible transitions are summarized on Illustration 9.

Illustration 9

5.4.4 Transitions to and from STATUS_PROCESSING

The status indicates that the device is currently busy with an operation. It can be time consuming operation and can result to any other status. The operation processing can be reached by any other status except STATUS_REMOVED. An example, offline device requires some data processing to become online. It will apply the statuses STATUS_OFFLINE, STATUS_PROCESSING and STATUS_ONLINE. Transitions to and from this status are:

- From STATUS_PROCESSING to STATUS_REMOVED – device is removed. The status can be set as a result of Device.remove() method call.
- From STATUS_PROCESSING to STATUS_ONLINE – Initial device data has been read. The device is available for operations.
- From STATUS_PROCESSING to STATUS_NOT_CONFIGURED – device has a pending configuration.
- From STATUS_PROCESSING to STATUS_OFFLINE – Online device is not accessible any more.
The status indicates that the device is currently not initialized. Some protocols don't provide device information right after the device is connected. The device can be initialized later when it's awakened. Not initialized device requires some data processing to become online. STATUS_PROCESSING is used as an intermediate status. Transitions to and from this status are:

- From STATUS_NOT_INITIALIZED to STATUS_REMOVED – device is removed. The status can be set as a result of Device.remove() method call.
- From STATUS_NOT_INITIALIZED to STATUS_PROCESSING – device data is processing.
- From STATUS_NOT_INITIALIZED to STATUS_NOT_CONFIGURED – That transition is not possible. device requires some data processing.
- From STATUS_NOT_INITIALIZED to STATUS_OFFLINE – device is not accessible any more.
- From STATUS_NOT_INITIALIZED to STATUS_ONLINE – That transition is not possible. Device requires some data processing to become online.
- To STATUS_NOT_INITIALIZED from STATUS_REMOVED – That transition is not possible. If device is removed, the service will be unregistered from the service registry.

Illustration 10

5.4.5 Transitions to and from STATUS_NOT_INITIALIZED

The possible transitions are summarized on Illustration 10.
To **STATUS_NOT_INITIALIZED** from **STATUS_PROCESSING** – device data is partially read.

To **STATUS_NOT_INITIALIZED** from **STATUS_NOT_CONFIGURED** – That transition is not possible. When device pending configuration is satisfied, the device requires additional data processing.

To **STATUS_NOT_INITIALIZED** from **STATUS_OFFLINE** – That transition is not possible. Device requires some data processing and then can become not initialized.

To **STATUS_NOT_INITIALIZED** from **STATUS_ONLINE** – That transition is not possible. Online device is initialized.

The possible transitions are summarized on Illustration 11.

### Illustration 11

#### 5.4.6 Transitions to and from STATUS_NOT_CONFIGURED

Indicates that the device is currently not configured. The device can require additional actions to become completely connected to the network. For example, a given device button has to be pushed. That status doesn't have transitions with **STATUS_NOT_INITIALIZED**, because some data processing is required. Transitions to and from this status are:

- From **STATUS_NOT_CONFIGURED** to **STATUS_REMOVED** – device is removed. The status can be set as a result of `Device.remove()` method call.
- From **STATUS_NOT_CONFIGURED** to **STATUS_PROCESSING** – device pending configuration is satisfied and some additional data processing is required.
- From **STATUS_NOT_CONFIGURED** to **STATUS_ONLINE** – device pending configuration is satisfied.
- From **STATUS_NOT_CONFIGURED** to **STATUS_OFFLINE** – device is not accessible any more.
- From **STATUS_NOT_CONFIGURED** to **STATUS_NOT_INITIALIZED** – That transition is not possible. When device pending configuration is satisfied, the device requires additional data processing.
- To **STATUS_NOT_CONFIGURED** from **STATUS_REMOVED** – That transition is not possible. If device is removed, the service will be unregistered from the service registry.
To STATUS_NOT_CONFIGURED from STATUS_PROCESSING – Initial device data has been read but there is a pending configuration.

To STATUS_NOT_CONFIGURED from STATUS_ONLINE – device has a pending configuration.

To STATUS_NOT_CONFIGURED from STATUS_OFFLINE – device is going to be online, but has a pending configuration.

To STATUS_NOT_CONFIGURED from STATUS_NOT_INITIALIZED – That transition is not possible. Device requires some data processing.

The possible transitions are summarized on Illustration 12.

**5.5 Functions**

The user applications can execute the device operations and manage the device properties. That control is realized with the help of Function services. The Function service can be registered in the service registry with those service properties:

- Function.SERVICE_UID – mandatory service property. The property value is the function unique identifier. The value type is java.lang.String. To simplify the unique identifier generation, the property value must follow the rule:
  
  function UID ::= device-id ':' function-id

  function UID – function unique identifier
  
  device-id – the value of the Device.SERVICE_UID Device service property

  function-id – function identifier in the scope of the device

  If the function is not bound to a device, the function unique identifier can be device independent.
• Function.SERVICE_TYPE – mandatory service property. The service property value contains the function type. For example, the sensor function can have different types like temperature or pressure etc. It's an optional property. The value type is java.lang.String.

Organizations that want to use function types that do not clash with OSGi Alliance defined types should prefix their types in own namespace.

• Function.SERVICE_VERSION – optional service property. The service property value contains the function version. That version can point to specific implementation version and vary in the different vendor implementations. The value type is java.lang.String.

• Function.SERVICE_DEVICE_UID – optional service property. The property value is the device identifier. The function belongs to this device. The value type is java.lang.String.

• Function.SERVICE_REFERENCE_UIDS – optional service property. The service property value contains the reference function unique identifiers. The value type is java.lang.String[]. It can be used to represent different relationships between the functions.

• Function.SERVICE_DESCRIPTION – optional service property. The property value is the function description. The value type is java.lang.String.

• Function.SERVICE_OPERATION_NAMES – optional service property. The property is missing when there are no function operations and property must be set when there are function operations. The property value is the function operation names. The value type is java.lang.String[]. It's not possible to exist two or more function operations with the same name i.e. the operation overloading is not allowed.

• Function.SERVICE_PROPERTY_NAMES – optional service property. The property is missing when there are no function properties and property must be set when there are function properties. The property value is the function property names. The value type is java.lang.String[]. It's not possible to exist two or more function properties with the same name.

The Function services are registered before the Device service. It's possible that Function.SERVICE_DEVICE_UID points to missing services at the moment of the registration. The reverse order is used when the services are unregistered. Device service is unregistered before the Function services.

Function service must be registered under the function class hierarchy. Other interfaces are not allowed. All classes from the function class hierarchy must participate as registration classes in the order from child to parent. The Function interface must be the last one in the list. For example, MeterV2 extends MeterV1 extends Function are function interfaces. If the implementation would like to provide MeterV2 functionality, the registration is: context.registerService(new String[]{MeterV2.class.getName(), MeterV1.class.getName(), Function.class.getName()}, this, regProps); MeterV2 is the last child in the class hierarchy and it's on the first position. MeterV1 is a parent of MeterV2 and child of Function. MeterV1 position is between MeterV2 and Function in the registration classes. If the implementation would like to provide MeterV1 functionality, the registration is: context.registerService(new String[]{MeterV1.class.getName(), Function.class.getName()}, this, regProps); If the implementation would like to mark that there is a function, but no specific function interface exists, the registration can be: context.registerService(new String[]{Function.class.getName()}, this, regProps); Note that such functions usually don't have operations and properties.

Some examples of not allowed registrations:

• context.registerService(new String[] {ManagedService.class.getName(), Function.class.getName()}, this, regProps); - ManagedService interface doesn't participate in a function class hierarchy.

• context.registerService(new String[] {MeterV1.class.getName()}, this, regProps); - Function interface is missing.
context.registerService(new String[] {MeterV1.class.getName(), Alarm.class.getName(), Function.class.getName()}, this, regProps); where MeterV1 extends Function and Alarm extends Function. - MeterV1 and Alarm are from different function class hierarchies.

That registration rule helps to the applications to find the supported function classes and to identify the metadata. Otherwise the function services can be accesses, but it's not clear which are the function classes and metadata.

5.5.1 Function Interface

Function is built by a set of properties and operations. The function can have unique identifier, type, version, description, link to the Device service and information about the reference functions. Function interface must be the base interface for all functions. If the device provider defines custom functions, all of them must extend Function interface. It provides a common access to the operations and properties meta data.

There are some general type rules, which unifies the access to the function data. They make easier the transfer over different protocols. All properties and operation arguments must use:

- Java primitive type or corresponding reference type.
- java.lang.String
- Java Beans, but their properties must use those rules. Java Beans are defined in JavaBeans specification [3].
- java.util.Map instances. The map keys can be any reference type of Java primitive types or java.lang.String. The values must use those rules.
- Arrays of defined types.

In order to provide common behavior, all functions must follow a set of common rules related to the implementation of their setters, getters, operations and events:

- The setter method must be executed synchronously. If the underlying protocol can return response to the setter call, it must be awaited. It simplifies the property value modifications and doesn't require asynchronous callback.
- The operation method must be executed synchronously. If the underlying protocol can return an operation confirmation or response, they must be awaited. It simplifies the operation execution and doesn't require asynchronous callback.
- The getter must return the last know cached property value. The device implementation is responsible to keep that value up to date. It'll speed up the applications when the function property values are collected. The same cached value can be shared between a few requests instead of a few calls to the real device.
- If a given function operation, getter or setter is not supported, java.lang.UnsupportedOperationException must be thrown. It indicates that function is partially supported.
- The function operations, getters and setters must not override java.lang.Object and this interface methods. For example:
  - hashCode() – it's java.lang.Object method and invalid function operation;
  - wait() – it's java.lang.Object method and invalid function operation;
  - getClass() – it's java.lang.Object method and invalid function getter;
  - getPropertyMetadata(String propertyName) – it's org.osgi.service.dal.Function method and invalid function getter.
5.5.2 Function Operations

Function operations are general callable units. They can perform a specific task on the device like turn on or turn off. They can be used by the applications to control the device. Operation names are available as a value of the service property `Function.SERVICE_OPERATION_NAMES`. The operations are identified by their names. It's not possible to exist two operations with the same name i.e. overloaded operations are not allowed or to override the property accessor methods. The operations are regular java methods. That implies that they have zero or more arguments and zero or one return value. The operation arguments and return value must follow the general type rules.

The operations can be optionally described with a set of meta data properties. Metadata is accessible with `Function.getOperationMetadata(String)` method. The result provides metadata about the operation, operation arguments and result value. Operation arguments and result value are using the same metadata as the function properties. The full details are defined in the next section.

5.5.3 Function Properties

Function properties are class fields. Their values can be read with getter methods and can be set with setter methods. The property names are available as a value of the service property `Function.SERVICE_PROPERTY_NAMES`. The properties are identified by their names. It's not possible to exist two properties with the same name.

The function properties must be integrated according to these rules:

- Getter methods must be available for all properties with `PropertyMetadata.PROPERTY_ACCESS_READABLE` access.
- Getter method must return a subclass of `FunctionData`.
- Setter methods must be available for all properties with `PropertyMetadata.PROPERTY_ACCESS_WRITABLE` access.
- Setter method must use `FunctionData` wrapped type. For example, there is `MyFunctionData` with timestamp, unit and `BigDecimal` value. The setter must accept as an argument the value of type `BigDecimal`.
- It's possible to have a second setter method, which accepts the value as a first argument and the unit as a second argument.
- No methods are required for properties with `PropertyMetadata.PROPERTY_ACCESS_EVENTABLE` access.

The accessor method names must be defined according JavaBeans specification [3].

The properties can be optionally described with a set of meta data properties. The property values can be collected with `Function.getPropertyMetadata(String)` method. The method result is `PropertyMetadata` with:

- Minimum value – available through `PropertyMetadata.getMinValue(String)`. The minimum value can be different for the different units.
- Maximum value – available through `PropertyMetadata.getMaxValue(String)`. The maximum value can be different for the different units.
- Enumeration of values – available through `PropertyMetadata.getEnumValues(String)`. The array of the possible values is sorted in increasing order according to the given unit.
- Resolution – available through `PropertyMetadata.getResolution(String)`. For example, if the range is [0, 100], the resolution can be 10. That's the different between two values in series. The resolution type depends on the property type. If the property is using data bean like `org.osgi.service.dal.functions.data.LevelData`, the resolution will be `BigDecimal`.
• Property access – available as a value in PropertyMetadata.getMetadata(String) result map. It's a bitmap of java.lang.Integer type and doesn't depend on the given unit. The access is available only for the function properties and it's missing for the operation arguments and result metadata. The bitmap can be any combination of:
  ◦ PropertyMetadata.PROPERTY_ACCESS_READABLE – Marks the property as a readable. Function must provide a getter method for this property according to JavaBeans specification [3]. Function operations must not be overridden by this getter method.
  ◦ PropertyMetadata.PROPERTY_ACCESS_WRITABLE – Marks the property as writable. Function must provide a setter method for this property according to JavaBeans specification [3]. Function operations must not be overridden by this setter method.
  ◦ PropertyMetadata.PROPERTY_ACCESS_EVENTABLE – Marks the property as eventable. Function must not provide special methods because of this access type. FunctionEvent is sent on property change. Note that the event can be sent when there is no value change.

• Unit - available as a value in PropertyMetadata.getMetadata() result map. The value contains the property supported units. The property value type is java.lang.String[]. Each unit must follow those rules:
  ◦ The International System of Units must be used where it's applicable. For example, kg for kilogram and km for kilometre.
  ◦ If the unit name matches to a Unicode symbol name, the Unicode symbol must be used. For example, the degree unit matches to the Unicode degree sign (°).
  ◦ If the unit name doesn't match to a Unicode symbol, the unit symbol must be built by Unicode Basic Latin block of characters, superscript and subscript characters. For example, watt per square metre steradian is built by W/(m² sr), where ² is Unicode superscript two.

If those rules cannot be applied to the unit symbol, custom rules are allowed. A set of predefined unit symbols are available in Units interface.

• Description – available as a value in PropertyMetadata.getMetadata() result map. The property value type is java.lang.String and specifies a user readable description. It doesn't depend on the given unit.

• Vendor custom properties – available as a value in PropertyMetadata.getMetadata() result map and can depend on the given unit.

5.5.4 Function Property Event

The eventable function properties can trigger a new event on each property value touch. It doesn't require a modification of the value. For example, the motion sensor can send a few events with no property value change when motion is detected and continued to be detected. The event must implement FunctionEvent interface. The event properties are:

• FunctionEvent.PROPERTY_FUNCTION_UID – the event source function unique identifier.
• FunctionEvent.PROPERTY_FUNCTION_PROPERTY_NAME – the property name.
• FunctionEvent.PROPERTY_FUNCTION_PROPERTY_VALUE – the property value.

For example, there is function with an eventable boolean property called “state”. When “state” value is changed to false, function implementation can post:

FunctionEvent {
    dal.function.UID=acme.function
6 Data Transfer Objects

DTOs are out of scope.

7 Javadoc
## Package Summary

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.service.dal</td>
<td>Device Package Version 1.0.</td>
<td>31</td>
</tr>
</tbody>
</table>
Package org.osgi.service.dal

Device Package Version 1.0.

See: Description

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>32</td>
</tr>
<tr>
<td>Function</td>
<td>46</td>
</tr>
<tr>
<td>OperationMetadata</td>
<td>58</td>
</tr>
<tr>
<td>PropertyMetadata</td>
<td>60</td>
</tr>
<tr>
<td>Units</td>
<td>64</td>
</tr>
</tbody>
</table>

Device represents the device in the OSGi service registry.

Function service provides specific device operations and properties.

OperationMetadata contains metadata about function operation.

PropertyMetadata contains metadata about function property or function operation parameter.

Units contains the most of the International System of Units unit symbols.

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevicePermission</td>
<td>43</td>
</tr>
<tr>
<td>FunctionData</td>
<td>51</td>
</tr>
<tr>
<td>FunctionEvent</td>
<td>55</td>
</tr>
</tbody>
</table>

DevicePermission is a bundle's authority to perform specific privileged administrative operations on the devices.

Abstract Function data wrapper.

Asynchronous event, which marks a function property value modification.

<table>
<thead>
<tr>
<th>Exception Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceException</td>
<td>39</td>
</tr>
</tbody>
</table>

DeviceException is a special IOException, which is thrown to indicate that there is a device operation fail.

Package org.osgi.service.dal Description

Device 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.dal; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

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

Represents the device in the OSGi service registry. Note that Device services are registered last. Before their registration, there is Function Services registration. The reverse order is used when the services are unregistered. Device services are unregistered first before Function services.

Field Summary

<table>
<thead>
<tr>
<th>String</th>
<th>DEVICE_CATEGORY</th>
<th>Constant for the value of the \texttt{org.osgi.service.device.Constants.DEVICE_CATEGORY} service property.</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>SERVICE_DESCRIPTION</td>
<td>The service property value contains the device description.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_DRIVER</td>
<td>The service property value contains the device driver name.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_FIRMWARE_VENDOR</td>
<td>The service property value contains the device firmware vendor.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_FIRMWARE_VERSION</td>
<td>The service property value contains the device firmware version.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_HARDWARE_VENDOR</td>
<td>The service property value contains the device hardware vendor.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_HARDWARE_VERSION</td>
<td>The service property value contains the device hardware version.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_MODEL</td>
<td>The service property value contains the device model.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_NAME</td>
<td>The service property value contains the device name.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_REFERENCE_UIDS</td>
<td>The service property value contains the reference device unique identifiers.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_SERIAL_NUMBER</td>
<td>The service property value contains the device serial number.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_STATUS</td>
<td>The service property value contains the device status.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_STATUS_DETAIL</td>
<td>The service property value contains the device status detail.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_TYPES</td>
<td>The service property value contains the device types like DVD, TV etc.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_UID</td>
<td>The service property value contains the device unique identifier.</td>
</tr>
<tr>
<td>Integer</td>
<td>STATUS_DETAIL_CONFIGURATION_NOT_APPLIED</td>
<td>Device status detail indicates that the device configuration is not applied.</td>
</tr>
<tr>
<td>Integer</td>
<td>STATUS_DETAIL_CONNECTING</td>
<td>Device status detail indicates that the device is currently connecting to the network.</td>
</tr>
<tr>
<td>Integer</td>
<td>STATUS_DETAIL_DEVICE_BROKEN</td>
<td>Device status detail indicates that the device is broken.</td>
</tr>
</tbody>
</table>
Device status detail indicates that the device communication is problematic.

Device status detail indicates that the device doesn't provide enough information and cannot be determined.

Device status detail indicates that the device is not accessible and further communication is not possible.

Device status detail indicates that the device cannot be configured.

Device status detail indicates that the device is in duty cycle.

Device status detail indicates that the device is currently in process of initialization.

Device status detail indicates that the device is leaving the network.

Device status indicates that the device is currently not configured.

Device status indicates that the device is currently not initialized.

Device status indicates that the device is currently not available for operations.

Device status indicates that the device is currently available for operations.

Device status indicates that the device is currently busy with an operation.

Device status indicates that the device is removed from the network.

Returns the current value of the specified property.

Removes this device.

Constant for the value of the org.osgi.service.device.Constants.DEVICE_CATEGORY service property. That category is used by all device services.

See Also:
org.osgi.service.device.Constants.DEVICE_CATEGORY

SERVICE_UID

public static final String SERVICE_UID = "dal.device.UID"
The service property value contains the device unique identifier. It's a mandatory property. The value type is `java.lang.String`. To simplify the unique identifier generation, the property value must follow the rule:

\[
\text{UID ::= driver-name ':' device-id}
\]

**UID** - device unique identifier

**driver-name** - the value of the `SERVICE_DRIVER` service property

**device-id** - device unique identifier in the scope of the driver

---

### `SERVICE_REFERENCE_UIDS`

```java
public static final String SERVICE_REFERENCE_UIDS = "dal.device.reference.UIDs"
```

The service property value contains the reference device unique identifiers. It's an optional property. The value type is `java.lang.String[]`. It can be used to represent different relationships between the devices. For example, the ZigBee controller can have a reference to the USB dongle.

---

### `SERVICE_DRIVER`

```java
public static final String SERVICE_DRIVER = "dal.device.driver"
```

The service property value contains the device driver name. For example, ZigBee, Z-Wave, Bluetooth etc. It's a mandatory property. The value type is `java.lang.String`.

---

### `SERVICE_NAME`

```java
public static final String SERVICE_NAME = "dal.device.name"
```

The service property value contains the device name. It's an optional property. The value type is `java.lang.String`.

---

### `SERVICE_STATUS`

```java
public static final String SERVICE_STATUS = "dal.device.status"
```

The service property value contains the device status. It's a mandatory property. The value type is `java.lang.Integer`. The possible values are:

- `STATUS_ONLINE`
- `STATUS_OFFLINE`
- `STATUS_REMOVED`
- `STATUS_PROCESSING`
- `STATUS_NOT_INITIALIZED`
- `STATUS_NOT_CONFIGURED`

---

### `SERVICE_STATUS_DETAIL`

```java
public static final String SERVICE_STATUS_DETAIL = "dal.device.status.detail"
```

The service property value contains the device status detail. It holds the reason for the current device status. It's an optional property. The value type is `java.lang.Integer`. There are two value categories:

- positive values i.e. > 0
- Those values contain details related to the current status. Examples: `STATUS_DETAIL_CONNECTING` and `STATUS_DETAIL_INITIALIZING`
- negative values i.e. 0
- Those values contain errors related to the current status. Examples: `STATUS_DETAIL_CONFIGURATION_NOT_APPLIED`, `STATUS_DETAIL_DEVICE_BROKEN` and `STATUS_DETAIL_DEVICE_COMMUNICATION_ERROR`
### Interface AbstractDevice

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_HARDWARE_VENDOR</td>
<td><code>dal.device.hardware.vendor</code></td>
<td>The service property value contains the device hardware vendor. It's an optional property. The value type is <code>java.lang.String</code>.</td>
</tr>
<tr>
<td>SERVICE_HARDWARE_VERSION</td>
<td><code>dal.device.hardware.version</code></td>
<td>The service property value contains the device hardware version. It's an optional property. The value type is <code>java.lang.String</code>.</td>
</tr>
<tr>
<td>SERVICE_FIRMWARE_VENDOR</td>
<td><code>dal.device.firmware.vendor</code></td>
<td>The service property value contains the device firmware vendor. It's an optional property. The value type is <code>java.lang.String</code>.</td>
</tr>
<tr>
<td>SERVICE_FIRMWARE_VERSION</td>
<td><code>dal.device.firmware.version</code></td>
<td>The service property value contains the device firmware version. It's an optional property. The value type is <code>java.lang.String</code>.</td>
</tr>
<tr>
<td>SERVICE_TYPES</td>
<td><code>dal.device.types</code></td>
<td>The service property value contains the device types like DVD, TV etc. It's an optional property. The value type is <code>java.lang.String[]</code>.</td>
</tr>
<tr>
<td>SERVICE_MODEL</td>
<td><code>dal.device.model</code></td>
<td>The service property value contains the device model. It's an optional property. The value type is <code>java.lang.String</code>.</td>
</tr>
<tr>
<td>SERVICE_SERIAL_NUMBER</td>
<td><code>dal.device.serial.number</code></td>
<td>The service property value contains the device serial number. It's an optional property. The value type is <code>java.lang.String</code>.</td>
</tr>
<tr>
<td>SERVICE_DESCRIPTION</td>
<td><code>dal.device.description</code></td>
<td>The service property value contains the device description. It's an optional property. The value type is <code>java.lang.String</code>.</td>
</tr>
<tr>
<td>STATUS_REMOVED</td>
<td></td>
<td>Device status indicates that the device is removed from the network. That status must be set as the last device status and after that the device service can be unregistered from the service registry. It can be used as a value of <code>SERVICE_STATUS</code> service property.</td>
</tr>
</tbody>
</table>
**Interface AbstractDevice**

**STATUS_OFFLINE**

```java
public static final Integer STATUS_OFFLINE
```

Device status indicates that the device is currently not available for operations. It can be used as a value of `SERVICE_STATUS` service property.

**STATUS_ONLINE**

```java
public static final Integer STATUS_ONLINE
```

Device status indicates that the device is currently available for operations. It can be used as a value of `SERVICE_STATUS` service property.

**STATUS_PROCESSING**

```java
public static final Integer STATUS_PROCESSING
```

Device status indicates that the device is currently busy with an operation. It can be used as a value of `SERVICE_STATUS` service property.

**STATUS_NOT_INITIALIZED**

```java
public static final Integer STATUS_NOT_INITIALIZED
```

Device status indicates that the device is currently not initialized. Some protocols don't provide device information right after the device is connected. The device can be initialized later when it's awakened. It can be used as a value of `SERVICE_STATUS` service property.

**STATUS_NOT_CONFIGURED**

```java
public static final Integer STATUS_NOT_CONFIGURED
```

Device status indicates that the device is currently not configured. The device can require additional actions to become completely connected to the network. It can be used as a value of `SERVICE_STATUS` service property.

**STATUS_DETAIL_CONNECTING**

```java
public static final Integer STATUS_DETAIL_CONNECTING
```

Device status detail indicates that the device is currently connecting to the network. It can be used as a value of `SERVICE_STATUS_DETAIL` service property. The device status must be `STATUS_PROCESSING`.

**STATUS_DETAIL_INITIALIZING**

```java
public static final Integer STATUS_DETAIL_INITIALIZING
```

Device status detail indicates that the device is currently in process of initialization. It can be used as a value of `SERVICE_STATUS_DETAIL` service property. The device status must be `STATUS_PROCESSING`.

**STATUS_DETAIL_REMOVING**

```java
public static final Integer STATUS_DETAIL_REMOVING
```

Device status detail indicates that the device is leaving the network. It can be used as a value of `SERVICE_STATUS_DETAIL` service property. The device status must be `STATUS_PROCESSING`.

**STATUS_DETAIL_CONFIGURATION_NOT_APPLIED**

```java
public static final Integer STATUS_DETAIL_CONFIGURATION_NOT_APPLIED
```

Device status detail indicates that the device configuration is not applied. It can be used as a value of `SERVICE_STATUS_DETAIL` service property. The device status must be `STATUS_NOT_CONFIGURED`.
### STATUS_DETAIL_DEVICE_BROKEN

public static final Integer STATUS_DETAIL_DEVICE_BROKEN

Device status detail indicates that the device is broken. It can be used as a value of `SERVICE_STATUS_DETAIL` service property. The device status must be `STATUS_OFFLINE`.

### STATUS_DETAIL_DEVICE_COMMUNICATION_ERROR

public static final Integer STATUS_DETAIL_DEVICE_COMMUNICATION_ERROR

Device status detail indicates that the device communication is problematic. It can be used as a value of `SERVICE_STATUS_DETAIL` service property. The device status must be `STATUS_ONLINE` or `STATUS_NOT_INITIALIZED`.

### STATUS_DETAIL_DEVICE_DATA_INSUFFICIENT

public static final Integer STATUS_DETAIL_DEVICE_DATA_INSUFFICIENT

Device status detail indicates that the device doesn't provide enough information and cannot be determined. It can be used as a value of `SERVICE_STATUS_DETAIL` service property. The device status must be `STATUS_NOT_INITIALIZED`.

### STATUS_DETAIL_DEVICE_NOT_ACCESSIBLE

public static final Integer STATUS_DETAIL_DEVICE_NOT_ACCESSIBLE

Device status detail indicates that the device is not accessible and further communication is not possible. It can be used as a value of `SERVICE_STATUS_DETAIL` service property. The device status must be `STATUS_OFFLINE`.

### STATUS_DETAIL_ERROR_APPLYING_CONFIGURATION

public static final Integer STATUS_DETAIL_ERROR_APPLYING_CONFIGURATION

Device status detail indicates that the device cannot be configured. It can be used as a value of `SERVICE_STATUS_DETAIL` service property. The device status must be `STATUS_NOT_CONFIGURED`.

### STATUS_DETAIL_IN_DUTY_CYCLE

public static final Integer STATUS_DETAIL_IN_DUTY_CYCLE

Device status detail indicates that the device is in duty cycle. It can be used as a value of `SERVICE_STATUS_DETAIL` service property. The device status must be `STATUS_OFFLINE`.

---

### Method Detail

getServiceProperty

Object getIServiceProperty(String propName)

Returns the current value of the specified property. The method will return the same value as `org.osgi.framework.ServiceReference.getProperty(String)` for the service reference of this device.

This method must continue to return property values after the device service has been unregistered.

**Parameters:**
- propName - The property name.

**Returns:**
- The property value or `null` if the property name cannot be mapped to a value.
Interface AbstractDevice

void remove()
throws DeviceException,
    UnsupportedOperationException,
    SecurityException,
    IllegalStateException

Removes this device. The method must synchronously remove the device from the device network.

Throw:

- **DeviceException** - If an operation error is available.
- **UnsupportedOperationException** - If the operation is not supported over this device.
- **SecurityException** - If the caller does not have the appropriate `DevicePermission[this device, DevicePermission.ACTION_REMOVE]` and the Java Runtime Environment supports permissions.
- **IllegalStateException** - If this device service object has already been unregistered.
Class DeviceException

org.osgi.service.dal

declares java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.io.IOException
       org.osgi.service.dal.DeviceException

All Implemented Interfaces:
  Serializable

public class DeviceException
  extends IOException

DeviceException is a special IOException, which is thrown to indicate that there is a device operation fail. The error reason can be located with getCode() method. The cause is available with getCause().

Field Summary

<table>
<thead>
<tr>
<th>static int</th>
<th>CODE COMMUNICATION_ERROR</th>
<th>An exception code indicates that there is an error in the communication.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE NO_DATA</td>
<td>An exception code indicates that the requested value is currently not available.</td>
<td></td>
</tr>
<tr>
<td>CODE NOT_INITIALIZED</td>
<td>An exception code indicates that the device is not initialized.</td>
<td></td>
</tr>
<tr>
<td>CODE TIMEOUT</td>
<td>An exception code indicates that there is expired timeout without any processing.</td>
<td></td>
</tr>
<tr>
<td>CODE UNKNOWN</td>
<td>An exception code indicates that the error is unknown.</td>
<td></td>
</tr>
</tbody>
</table>

Constructor Summary

| DeviceException() | Construct a new device exception with null message. |
| DeviceException(String message) | Constructs a new device exception with the given message. |
| DeviceException(String message, Throwable cause) | Constructs a new device exception with the given message and cause. |
| DeviceException(String message, Throwable cause, int code) | Constructs a new device exception with the given message, cause and code. |

Method Summary

| Throwable getCause() | Returns the cause for this exception or null if the cause is missing. |
| int getCode() | Returns the exception error code. |
| void printStackTrace() | Prints the exception stack trace to the standard error stream. |
| void printStackTrace(PrintStream s) | Prints the exception stack trace to the given stream. |
void printStackTrace(PrintWriter s)  
Prints the exception stack trace to the given writer.

Field Detail

CODE_UNKNOWN
public static final int CODE_UNKNOWN = 1

An exception code indicates that the error is unknown.

CODE_COMMUNICATION_ERROR
public static final int CODE_COMMUNICATION_ERROR = 2

An exception code indicates that there is an error in the communication.

CODE_TIMEOUT
public static final int CODE_TIMEOUT = 3

An exception code indicates that there is expired timeout without any processing.

CODE_NOT_INITIALIZED
public static final int CODE_NOT_INITIALIZED = 4

An exception code indicates that the device is not initialized. The device status is Device.STATUS_NOT_INITIALIZED or Device.STATUS_PROCESSING.

CODE_NO_DATA
public static final int CODE_NO_DATA = 5

An exception code indicates that the requested value is currently not available.

Constructor Detail

DeviceException
public DeviceException()

Construct a new device exception with null message. The cause is not initialized and the exception code is set to CODE_UNKNOWN.

DeviceException
public DeviceException(String message)

Constructs a new device exception with the given message. The cause is not initialized and the exception code is set to CODE_UNKNOWN.

Parameters:
message - The exception message.

DeviceException
public DeviceException(String message, Throwable cause)
Constructs a new device exception with the given message and cause. The exception code is set to `CODE_UNKNOWN`.

**Parameters:**
- `message` - The exception message.
- `cause` - The exception cause.

### DeviceException

```java
public DeviceException(String message, Throwable cause, int code)
```

Constructs a new device exception with the given message, cause and code.

**Parameters:**
- `message` - The exception message.
- `cause` - The exception cause.
- `code` - The exception code.

### Method Detail

#### getCode

```java
public int getCode()
```

Returns the exception error code. It indicates the reason for this exception.

**Returns:**
- An exception code.

#### getCause

```java
public Throwable getCause()
```

Returns the cause for this exception or `null` if the cause is missing. The cause can be protocol specific exception with an appropriate message and error code.

**Overrides:**
- `getCause in class Throwable`

**Returns:**
- An throwable cause.

#### printStackTrace

```java
public void printStackTrace()
```

Prints the exception stack trace to the standard error stream.

**Overrides:**
- `printStackTrace in class Throwable`

**See Also:**
- `Throwable.printStackTrace()`
Interface AbstractDeviceAdmin

printStackTrace

public void printStackTrace(PrintStream s)

Prints the exception stack trace to the given stream.

Overrides:
printStackTrace in class Throwable

Parameters:
s - The stream used for the output.

See Also:
Throwable.printStackTrace(java.io.PrintStream)

printStackTrace

public void printStackTrace(PrintWriter s)

Prints the exception stack trace to the given writer.

Overrides:
printStackTrace in class Throwable

Parameters:
s - The writer used for the output.

See Also:
Throwable.printStackTrace(java.io.PrintWriter)
Class AbstractDeviceException

org.osgi.service.dal

java.lang.Object
  java.security.Permission
    java.security.BasicPermission
      org.osgi.service.dal.DevicePermission

All Implemented Interfaces:
  Guard, Serializable

final public class DevicePermission
extends BasicPermission

A bundle's authority to perform specific privileged administrative operations on the devices. The method Device.remove() is protected with ACTION_REMOVE permission action.

The name of the permission is a filter based. See OSGi Core Specification, Filter Based Permissions. The filter gives an access to all device service properties. The service property names are case insensitive. The filter attribute names are processed in a case insensitive manner.

Field Summary

<table>
<thead>
<tr>
<th>static String ACTION_REMOVE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A permission action to remove the device.</td>
<td>44</td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>DevicePermission(String filter, String action)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates a new DevicePermission with the given filter and actions.</td>
<td>44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DevicePermission(Device device, String action)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates a new DevicePermission with the given device and actions.</td>
<td>44</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>boolean equals(Object obj)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two DevicePermission instances are equal if:</td>
<td>44</td>
</tr>
<tr>
<td>□ represents the same filter and actions</td>
<td></td>
</tr>
<tr>
<td>□ represents the same device and actions</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String getActions()</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the canonical string representation of ACTION_REMOVE action.</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int hashCode()</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the hash code value for this object.</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean implies(Permission p)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determines if the specified permission is implied by this object.</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PermissionCollection newPermissionCollection()</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns a new PermissionCollection suitable for storing DevicePermission instances.</td>
<td>45</td>
</tr>
</tbody>
</table>
Class AbstractDeviceException

Field Detail

ACTION_REMOVE

public static final String ACTION_REMOVE = "remove"

A permission action to remove the device.

Constructor Detail

DevicePermission

public DevicePermission(String filter, String action)

Creates a new DevicePermission with the given filter and actions. The constructor must only be used to create a permission that is going to be checked.

An filter example: (dal.device.hardware.vendor=acme)

An action list example: property, remove

Parameters:
  filter - A filter expression that can use any device service property. The filter attribute names are processed in a case insensitive manner. A special value of *** can be used to match all devices.
  action - ACTION_REMOVE action.

Throws:
  IllegalArgumentException - If the filter syntax is not correct or invalid actions are specified.

DevicePermission

public DevicePermission(Device device, String action)

Creates a new DevicePermission with the given device and actions. The permission must be used for the security checks like:

securityManager.checkPermission(new DevicePermission(this, "remove"));. The permissions constructed by this constructor must not be added to the DevicePermission permission collections.

Parameters:
  device - The permission device.
  action - ACTION_REMOVE action.

Method Detail

equals

public boolean equals(Object obj)

Two DevicePermission instances are equal if:

☑ represents the same filter and actions
☑ represents the same device and actions

Overrides:
  equals in class BasicPermission
**Class AbstractDeviceException**

**Parameters:**
- obj - The object being compared for equality with this object.

**Returns:**
- true if two permissions are equal, false otherwise.

**hashCode**

```java
public int hashCode()
```

Returns the hash code value for this object.

**Overrides:**
- `hashCode` in class `BasicPermission`

**Returns:**
- Hash code value for this object.

**getActions**

```java
public String getActions()
```

Returns the canonical string representation of `ACTION_REMOVE` action.

**Overrides:**
- `getActions` in class `BasicPermission`

**Returns:**
- The canonical string representation of the actions.

**implies**

```java
public boolean implies(Permission p)
```

Determines if the specified permission is implied by this object. The method will throw an exception if the specified permission was not constructed by `DevicePermission(Device, String)`. Returns true if the specified permission is a `DevicePermission` and this permission filter matches the specified permission device properties.

**Overrides:**
- `implies` in class `BasicPermission`

**Parameters:**
- p - The permission to be implied. It must be constructed by `DevicePermission(Device, String)`.

**Returns:**
- true if the specified permission is implied by this permission, false otherwise.

**Throws:**
- `IllegalArgumentException` - If the specified permission is not constructed by `DevicePermission(Device, String)`.

**newPermissionCollection**

```java
public PermissionCollection newPermissionCollection()
```

Returns a new `PermissionCollection` suitable for storing `DevicePermission` instances.

**Overrides:**
- `newPermissionCollection` in class `BasicPermission`

**Returns:**
- A new `PermissionCollection` instance.
public interface Function

Function service provides specific device operations and properties. Each function service must implement this interface. In addition to this interface, the implementation can provide own:

- properties;
- operations.

The function service can be registered in the service registry with those service properties:

- SERVICE_UID - mandatory service property. The property value contains the function unique identifier.
- SERVICE_DEVICE_UID - optional service property. The property value is the Functional Device identifiers.
- SERVICE_REFERENCE_UIDS - optional service property. The property value contains the reference function unique identifiers.
- SERVICE_TYPE - mandatory service property. The property value is the function type.
- SERVICE_VERSION - optional service property. The property value contains the function version.
- SERVICE_DESCRIPTION - optional service property. The property value is the function description.
- SERVICE_OPERATION_NAMES - optional service property. The property is missing when there are no function operations and property must be set when there are function operations. The property value is the function operation names.
- SERVICE_PROPERTY_NAMES - optional service property. The property is missing when there are no function properties and property must be set when there are function properties. The property value is the function property names.

The Function services are registered before the Device services. It's possible that SERVICE_DEVICE_UID point to missing services at the moment of the registration. The reverse order is used when the services are unregistered. Function services are unregistered last after Device services.

Function service must be registered under the function class hierarchy. Other interfaces are not allowed. All classes from the function class hierarchy must participate as registration classes in the order from child to parent. The Function interface must be the last one in the list. For example, MeterV2 extends MeterV1 extends Function are function interfaces. If the implementation would like to provide MeterV2 functionality, the registration is:

```
context.registerService(new String[] {MeterV2.class.getName(), MeterV1.class.getName(),
Function.class.getName()}, this, regProps);
```

MeterV2 is the last child in the class hierarchy and it's on the first position. MeterV1 is a parent of MeterV2 and child of Function. MeterV1 position is between MeterV2 and Function in the registration classes. If the implementation would like to provide MeterV1 functionality, the registration is:

```
context.registerService(new String[] {MeterV1.class.getName(), Function.class.getName()}, this, regProps);
```

If the implementation would like to mark that there is a function, but no specific function interface exists, the registration can be:

```
context.registerService(new String[] {Function.class.getName()}, this, regProps);
```

Note that such functions usually don't have operations and properties.

Some examples of not allowed registrations:

- context.registerService(new String[] {ManagedService.class.getName(),
Function.class.getName()}, this, regProps); - ManagedService Interface doesn't participate in a function class hierarchy.
- context.registerService(new String[] {MeterV1.class.getName()}, this, regProps); - Function Interface is missing.
- context.registerService(new String[] {MeterV1.class.getName(), Alarm.class.getName(),
Function.class.getName()}, this, regProps); - where MeterV1 extends Function and Alarm extends Function. MeterV1 and Alarm are from different function class hierarchies.
That registration rule helps to the applications to find the supported function classes and to identify the metadata. Otherwise the function services can be accesses, but it's not clear which are the function classes and metadata.

The function properties must be integrated according to these rules:

- Getter methods must be available for all properties with `PropertyMetadata.PROPERTY_ACCESS_READABLE` access.
- Getter method must return a subclass of `FunctionData`.
- Setter methods must be available for all properties with `PropertyMetadata.PROPERTY_ACCESS_WRITABLE` access.
- Setter method must use `FunctionData` wrapped type. For example, there is `MyFunctionData` with timestamp, unit and `BigDecimal` value. The setter must accept as an argument the value of type `BigDecimal`.
- It's possible to have a second setter method, which accepts the value as a first argument and the unit as a second argument.
- No methods are required for properties with `PropertyMetadata.PROPERTY_ACCESS_EVENTABLE` access.

The accessor method names must be defined according JavaBeans specification.

The function operations are java methods, which cannot override the property accessor methods. They can have zero or more parameters and zero or one return value.

Operation arguments and function properties are restricted by the same set of rules. The data type can be one of the following types:

- Java primitive type or corresponding reference type.
- `java.lang.String`.
- Beans, but the beans properties must use those rules. Java Beans are defined in JavaBeans specification.
- `java.util.Map`s. The keys can be any reference type of Java primitive types or `java.lang.String`. The values must use those rules.
- Arrays of defined types.

The properties metadata is accessible with `getPropertyMetadata(String)`. The operations metadata is accessible with `getOperationMetadata(String)`. In order to provide common behavior, all functions must follow a set of common rules related to the implementation of their setters, getters, operations and events:

- The setter method must be executed synchronously. If the underlying protocol can return response to the setter call, it must be awaited. It simplifies the property value modifications and doesn't require asynchronous callback.
- The operation method must be executed synchronously. If the underlying protocol can return an operation confirmation or response, they must be awaited. It simplifies the operation execution and doesn't require asynchronous callback.
- The getter must return the last know cached property value. The device implementation is responsible to keep that value up to date. It'll speed up the applications when the function property values are collected. The same cached value can be shared between a few requests instead of a few calls to the real device.
- If a given function operation, getter or setter is not supported, `java.lang.UnsupportedOperationException` must be thrown. It indicates that function is partially supported.
- The function operations, getters and setters must not override `java.lang.Object` and this interface methods.

### Field Summary

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>SERVICE_DESCRIPTION</td>
<td>The service property value contains the function description.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_DEVICE_UID</td>
<td>The service property value contains the device unique identifier.</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_OPERATION_NAMES</td>
<td>The service property value contains the function operation names.</td>
</tr>
</tbody>
</table>
Interface BaseDevice

<table>
<thead>
<tr>
<th>String SERVICE_PROPERTY_NAMES</th>
<th>The service property value contains the function property names.</th>
<th>49</th>
</tr>
</thead>
<tbody>
<tr>
<td>String SERVICE_REFERENCE_UIDS</td>
<td>The service property value contains the reference function unique identifiers.</td>
<td>49</td>
</tr>
<tr>
<td>String SERVICE_TYPE</td>
<td>The service property value contains the function type.</td>
<td>48</td>
</tr>
<tr>
<td>String SERVICE_UID</td>
<td>The service property value contains the function unique identifier.</td>
<td>48</td>
</tr>
<tr>
<td>String SERVICE_VERSION</td>
<td>The service property value contains the function version.</td>
<td>48</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getOperationMetadata(String)</td>
<td>Provides metadata about the function operation.</td>
<td>50</td>
</tr>
<tr>
<td>getPropertyMetadata(String)</td>
<td>Provides metadata about the function property specified with the name argument.</td>
<td>49</td>
</tr>
<tr>
<td>getServiceProperty(String)</td>
<td>Returns the current value of the specified property.</td>
<td>50</td>
</tr>
</tbody>
</table>

### Field Detail

**SERVICE_UID**

```
public static final String SERVICE_UID = "dal.function.UID"
```

The service property value contains the function unique identifier. It's a mandatory property. The value type is `java.lang.String`. To simplify the unique identifier generation, the property value must follow the rule:

```
function UID ::= device-id ':' function-id
```

- **function UID**: function unique identifier
- **device-id**: the value of the `Device.SERVICE_UID` Device service property
- **function-id**: function identifier in the scope of the device

If the function is not bound to a device, the function unique identifier can be device independent.

**SERVICE_TYPE**

```
public static final String SERVICE_TYPE = "dal.function.type"
```

The service property value contains the function type. It's an optional property. For example, the sensor function can have different types like temperature or pressure etc. The value type is `java.lang.String`.

Organizations that want to use function types that do not clash with OSGi Alliance defined types should prefix their types in own namespace.

The type doesn't mandate specific function interface. It can be used with different functions.

**SERVICE_VERSION**

```
public static final String SERVICE_VERSION = "dal.function.version"
```


The service property value contains the function version. That version can point to specific implementation version and vary in the different vendor implementations. It's an optional property. The value type is java.lang.String.

**SERVICE_DEVICE_UID**

```java
public static final String SERVICE_DEVICE_UID = "dal.function.device.UID"
```

The service property value contains the device unique identifier. The function belongs to this device. It's an optional property. The value type is java.lang.String.

**SERVICE_REFERENCE_UIDS**

```java
public static final String SERVICE_REFERENCE_UIDS = "dal.function.reference.UIDs"
```

The service property value contains the reference function unique identifiers. It's an optional property. The value type is java.lang.String[]. It can be used to represent different relationships between the functions.

**SERVICE_DESCRIPTION**

```java
public static final String SERVICE_DESCRIPTION = "dal.function.description"
```

The service property value contains the function description. It's an optional property. The value type is java.lang.String.

**SERVICE_OPERATION_NAMES**

```java
public static final String SERVICE_OPERATION_NAMES = "dal.function.operation.names"
```

The service property value contains the function operation names. It's an optional property. The property is missing when there are no function operations and property must be set when there are function operations. The value type is java.lang.String[]. It's not possible to exist two or more function operations with the same name i.e. the operation overloading is not allowed.

**SERVICE_PROPERTY_NAMES**

```java
public static final String SERVICE_PROPERTY_NAMES = "dal.function.property.names"
```

The service property value contains the function property names. It's an optional property. The property is missing when there are no function properties and property must be set when there are function properties. The value type is java.lang.String[]. It's not possible to exist two or more function properties with the same name.

### Method Detail

**getPropertyMetadata**

```java
PropertyMetadata getPropertyMetadata(String propertyName)
throws IllegalArgumentException
```

Provides metadata about the function property specified with the name argument.

This method must continue to return the property metadata after the function service has been unregistered.

**Parameters:**
- `propertyName`: The function property name, which metadata is requested.

**Returns:**
- The property metadata for the given property name. `null` if the property metadata is not supported.

**Throws:**
- `IllegalArgumentException` - If the function property with the specified name is not supported.
Interface BaseDevice

getOperationMetadata

OperationMetadata getOperationMetadata(String operationName) throws IllegalArgumentException

Provides metadata about the function operation.

This method must continue to return the operation metadata after the function service has been unregistered.

Parameters:
  operationName - The function operation name, which metadata is requested.

Returns:
  The operation metadata for the given operation name. null if the operation metadata is not supported.

Throws:
  IllegalArgumentException - If the function operation with the specified name is not supported.

getServiceProperty

Object getServiceProperty(String propName)

Returns the current value of the specified property. The method will return the same value as
org.osgi.framework.ServiceReference.getProperty(String) for the service reference of this function.

This method must continue to return property values after the device function service has been unregistered.

Parameters:
  propName - The property name.

Returns:
  The property value or null if the property name cannot be mapped to a value.
Abstract Function data wrapper. A subclass must be used for an access to the property values by all functions. It takes care about the timestamp and additional metadata. The subclasses are responsible to provide concrete value and unit if required.

The subclass is responsible to provide correct implementation of `Comparable.compareTo(Object)` method.

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FIELD_METADATA</code></td>
<td><code>String</code></td>
<td>Represents the metadata field name.</td>
<td>52</td>
</tr>
<tr>
<td><code>FIELD_TIMESTAMP</code></td>
<td><code>String</code></td>
<td>Represents the timestamp field name.</td>
<td>52</td>
</tr>
<tr>
<td><code>META_INFO_DESCRIPTION</code></td>
<td><code>String</code></td>
<td>Metadata key, which value represents the data description.</td>
<td>52</td>
</tr>
<tr>
<td><code>metadata</code></td>
<td><code>Map</code></td>
<td>Contains FunctionData metadata.</td>
<td>52</td>
</tr>
<tr>
<td><code>timestamp</code></td>
<td><code>long</code></td>
<td>Contains FunctionData timestamp.</td>
<td>52</td>
</tr>
</tbody>
</table>

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FunctionData(Map fields)</code></td>
<td>Constructs new FunctionData instance with the specified field values.</td>
<td>52</td>
</tr>
<tr>
<td><code>FunctionData(long timestamp, Map metadata)</code></td>
<td>Constructs new FunctionData instance with the specified arguments.</td>
<td>53</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>equals</code> (Object other)</td>
<td>Two FunctionData instances are equal if their metadata and timestamp are equivalent.</td>
<td>53</td>
</tr>
<tr>
<td><code>getMetadata()</code></td>
<td>Returns FunctionData metadata.</td>
<td>53</td>
</tr>
<tr>
<td><code>getTimestamp()</code></td>
<td>Returns FunctionData timestamp.</td>
<td>53</td>
</tr>
<tr>
<td><code>hashCode()</code></td>
<td>Returns the hash code of this FunctionData.</td>
<td>53</td>
</tr>
</tbody>
</table>
Field Detail

FIELD_TIMESTAMP

public static final String FIELD_TIMESTAMP = "timestamp"

Represents the timestamp field name. The field value is available with `timestamp` and `getTimestamp()`. The field type is `long`. The constant can be used as a key to `FunctionData(Map)`.

FIELD_METADATA

public static final String FIELD_METADATA = "metadata"

Represents the metadata field name. The field value is available with `metadata` and `getMetadata()`. The field type is `Map`. The constant can be used as a key to `FunctionData(Map)`.

META_INFO_DESCRIPTION

public static final String META_INFO_DESCRIPTION = "description"

Metadata key, which value represents the data description. The property value type is `java.lang.String`.

timestamp

public final long timestamp

Contains `FunctionData` `timestamp`. The timestamp is the difference between the value collecting time and midnight, January 1, 1970 UTC. It's measured in milliseconds. The device driver is responsible to generate that value when the value is received from the device. `Long.MIN_VALUE` value means no timestamp.

metadata

public final Map metadata

Contains `FunctionData` `metadata`. It's dynamic metadata related only to this specific value. Possible keys:

- META_INFO_DESCRIPTION
- custom key

Constructor Detail

FunctionData

public FunctionData(Map fields)

Constructs new `FunctionData` instance with the specified field values. The map keys must match to the field names. The map values will be assigned to the appropriate class fields. For example, the maps can be: {"timestamp"=Long(1384440775495)}. That map will initialize the `FIELD_TIMESTAMP` field with 1384440775495. If timestamp is missing, `Long.MIN_VALUE` is used.

`FIELD_TIMESTAMP` field value type must be `Long`. `FIELD_METADATA` field value type must be `Map`.

Parameters:

- `fields` - Contains the new `FunctionData` instance field values.

Throws:

- `ClassCastException` - If the field value types are not expected.
- `NullPointerException` - If the fields map is null.
FunctionData

```java
public FunctionData(long timestamp,
                     Map metadata)
```

Constructs new FunctionData instance with the specified arguments.

**Parameters:**
- `timestamp` - The data timestamp.
- `metadata` - The data metadata.

### Method Detail

#### getTimestamp

```java
public long getTimestamp()
```

Returns FunctionData timestamp. The timestamp is the difference between the value collecting time and midnight, January 1, 1970 UTC. It's measured in milliseconds. The device driver is responsible to generate that value when the value is received from the device. `Long.MIN_VALUE` value means no timestamp.

**Returns:**
FunctionData `timestamp`.

#### getMetadata

```java
public Map getMetadata()
```

Returns FunctionData `metadata`. It's dynamic metadata related only to this specific value. Possible keys:

- `META_INFO_DESCRIPTION`
- custom key

**Returns:**
FunctionData `metadata` or `null` is there is no metadata.

#### equals

```java
public boolean equals(Object other)
```

Two FunctionData instances are equal if their metadata and timestamp are equivalent.

**Overrides:**
 equals in class Object

**Parameters:**
- `other` - The other instance to compare. It must be of FunctionData type.

**Returns:**
true if this instance and argument have equivalent metadata and timestamp, false otherwise.

**See Also:**
Object.equals(java.lang.Object)

#### hashCode

```java
public int hashCode()
```

Returns the hash code of this FunctionData.
**Overrides:**

hashCode in class Object

**Returns:**

FunctionData hash code.

**See Also:**

Object.hashCode()
Class FunctionEvent

org.osgi.service.dal

text
java.lang.Object

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static String EVENT_CLASS</strong></td>
</tr>
<tr>
<td>Represents the event class.</td>
</tr>
<tr>
<td>Page 56</td>
</tr>
<tr>
<td><strong>static String EVENT_PACKAGE</strong></td>
</tr>
<tr>
<td>Represents the event package.</td>
</tr>
<tr>
<td>Page 56</td>
</tr>
<tr>
<td><strong>static String PROPERTY_FUNCTION_PROPERTY_NAME</strong></td>
</tr>
<tr>
<td>Represents an event property key for the function property name.</td>
</tr>
<tr>
<td>Page 56</td>
</tr>
<tr>
<td><strong>static String PROPERTY_FUNCTION_PROPERTY_VALUE</strong></td>
</tr>
<tr>
<td>Represents an event property key for the function property value.</td>
</tr>
<tr>
<td>Page 56</td>
</tr>
<tr>
<td><strong>static String PROPERTY_FUNCTION_UID</strong></td>
</tr>
<tr>
<td>Represents an event property key for function UID.</td>
</tr>
<tr>
<td>Page 56</td>
</tr>
<tr>
<td><strong>static String TOPIC_PROPERTY_CHANGED</strong></td>
</tr>
<tr>
<td>Represents the event topic for the function property changed.</td>
</tr>
<tr>
<td>Page 56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FunctionEvent (String topic, String funtionUID, String propName, FunctionData propValue)</strong></td>
</tr>
<tr>
<td>Constructs a new event with the specified topic, function UID, property name and property value.</td>
</tr>
<tr>
<td>Page 57</td>
</tr>
<tr>
<td><strong>FunctionEvent (String topic, Dictionary properties)</strong></td>
</tr>
<tr>
<td>Constructs a new event with the specified topic and properties.</td>
</tr>
<tr>
<td>Page 56</td>
</tr>
<tr>
<td><strong>FunctionEvent (String topic, Map properties)</strong></td>
</tr>
<tr>
<td>Constructs a new event with the specified topic and properties.</td>
</tr>
<tr>
<td>Page 57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String getFunctionPropertyName ()</strong></td>
</tr>
<tr>
<td>Returns the property name.</td>
</tr>
<tr>
<td>Page 57</td>
</tr>
<tr>
<td><strong>FunctionData getFunctionPropertyValue ()</strong></td>
</tr>
<tr>
<td>Returns the property value.</td>
</tr>
<tr>
<td>Page 57</td>
</tr>
<tr>
<td><strong>String getFunctionUID ()</strong></td>
</tr>
<tr>
<td>Returns the property value change source function identifier.</td>
</tr>
<tr>
<td>Page 57</td>
</tr>
</tbody>
</table>

Asynchronous event, which marks a function property value modification. The event can be triggered when there is a new property value, but it's possible to have events in series with no value change. The event properties must contain:

- **PROPERTY_FUNCTION_UID** - the event source function unique identifier.
- **PROPERTY_FUNCTION_PROPERTY_NAME** - the property name.
- **PROPERTY_FUNCTION_PROPERTY_VALUE** - the property value. The property value type must be a subclass of FunctionData.
Methods inherited from class org.osgi.service.event.Event
equals, getProperty, getPropertyNames, getTopic, hashCode, matches, toString

Field Detail

EVENT_PACKAGE
public static final String EVENT_PACKAGE = "org/osgi/service/dal/")

Represents the event package. That constant can be useful for the event handlers depending on the event filters.

EVENT_CLASS
public static final String EVENT_CLASS = "org/osgi/service/dal/FunctionEvent/"

Represents the event class. That constant can be useful for the event handlers depending on the event filters.

TOPIC_PROPERTY_CHANGED
public static final String TOPIC_PROPERTY_CHANGED =
"org/osgi/service/dal/FunctionEvent/PROPERTY_CHANGED"

Represents the event topic for the function property changed.

PROPERTY_FUNCTION_UID
public static final String PROPERTY_FUNCTION_UID = "dal.function.UID"

Represents an event property key for function UID. The property value type is java.lang.String. The value represents the property value change source function identifier.

PROPERTY_FUNCTION_PROPERTY_NAME
public static final String PROPERTY_FUNCTION_PROPERTY_NAME = "dal.function.property.name"

Represents an event property key for the function property name. The property value type is java.lang.String. The value represents the property name.

PROPERTY_FUNCTION_PROPERTY_VALUE
public static final String PROPERTY_FUNCTION_PROPERTY_VALUE = "dal.function.property.value"

Represents an event property key for the function property value. The property value type is a subclass of FunctionData. The value represents the property value.

Constructor Detail

FunctionEvent
public FunctionEvent(String topic,
Dictionary properties)

Constructs a new event with the specified topic and properties.

Parameters:
  topic - The event topic.
  properties - The event properties.
**Interface FunctionalDevice**

---

### FunctionEvent

**public FunctionEvent(String topic, Map properties)**

Constructs a new event with the specified topic and properties.

**Parameters:**
- `topic` - The event topic.
- `properties` - The event properties.

---

**public FunctionEvent(String topic, String funtionUID, String propName, FunctionData propValue)**

Constructs a new event with the specified topic, function UID, property name and property value.

**Parameters:**
- `topic` - The event topic.
- `funtionUID` - The event source function UID.
- `propName` - The event source property name.
- `propValue` - The event source property value.

---

#### Method Detail

---

**getFunctionUID**

**public String getFunctionUID()**

Returns the property value change source function identifier. The value is same as the value of `PROPERTY_FUNCTION_UID` property.

**Returns:**
- The property value change source function.

---

**getFunctionPropertyName**

**public String getFunctionPropertyName()**

Returns the property name. The value is same as the value of `PROPERTY_FUNCTION_PROPERTY_NAME`.

**Returns:**
- The property name.

---

**getFunctionPropertyValue**

**public FunctionData getFunctionPropertyValue()**

Returns the property value. The value is same as the value of `PROPERTY_FUNCTION_PROPERTY_VALUE`.

**Returns:**
- The property value.
public interface OperationMetadata

Contains metadata about function operation.

See Also:
Function, PropertyMetadata

Field Summary

<table>
<thead>
<tr>
<th>Page</th>
<th>String</th>
<th>META_INFO_DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Metadata key, which value represents the operation description.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Page</th>
<th>Map</th>
<th>getMetadata()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Returns metadata about the function operation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>PropertyMetadata</th>
<th>getParametersMetadata()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Returns metadata about the operation parameters or null if no such metadata is available.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>PropertyMetadata</th>
<th>getReturnValueMetadata()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Returns metadata about the operation return value or null if no such metadata is available.</td>
</tr>
</tbody>
</table>

Field Detail

META_INFO_DESCRIPTION
public static final String META_INFO_DESCRIPTION = "description"

Metadata key, which value represents the operation description. The property value type is java.lang.String.

Method Detail

getMetadata
Map getMetadata()

Returns metadata about the function operation. The keys of the java.util.Map result must be of java.lang.String type. Possible keys:

- META_INFO_DESCRIPTION
- custom key

Returns:
The operation metadata or null if no such metadata is available.

getReturnValueMetadata
PropertyMetadata getReturnValueMetadata()


Returns metadata about the operation return value or `null` if no such metadata is available.

**Returns:**
Operation return value metadata.

---

### `getParametersMetadata`

```
PropertyMetadata[] getParametersMetadata()
```

Returns metadata about the operation parameters or `null` if no such metadata is available.

**Returns:**
Operation parameters metadata.
public interface PropertyMetadata

Contains metadata about function property or function operation parameter. The access to the function properties is a bitmap value of PROPERTY_ACCESS metadata key. Function properties can be accessed in three ways. Any combinations between them are possible:

- **PROPERTY_ACCESS_READABLE** - available for all properties, which can be read. Function must provide a getter method for an access to the property value.
- **PROPERTY_ACCESS_WRITABLE** - available for all properties, which can be modified. Function must provide a setter method for a modification of the property value.
- **PROPERTY_ACCESS_EVENTABLE** - available for all properties, which can report the property value. FunctionEvents are sent on property change.

See Also:
- Function, PropertyMetadata

---

**Field Summary**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>String</td>
<td>Metadata key, which value represents the property description.</td>
<td>61</td>
</tr>
<tr>
<td>PROPERTY_ACCESS</td>
<td>String</td>
<td>Metadata key, which value represents the access to the function property.</td>
<td>61</td>
</tr>
<tr>
<td>PROPERTY_ACCESS_EVENTABLE</td>
<td>int</td>
<td>Marks the eventable function properties.</td>
<td>61</td>
</tr>
<tr>
<td>PROPERTY_ACCESS_READABLE</td>
<td>int</td>
<td>Marks the readable function properties.</td>
<td>61</td>
</tr>
<tr>
<td>PROPERTY_ACCESS_WRITABLE</td>
<td>int</td>
<td>Marks the writable function properties.</td>
<td>61</td>
</tr>
<tr>
<td>UNITS</td>
<td>String</td>
<td>Metadata key, which value represents the property supported units.</td>
<td>61</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FunctionData[] getEnumValues(String unit)</td>
<td></td>
<td>Returns the property possible values according to the specified unit.</td>
<td>62</td>
</tr>
<tr>
<td>FunctionData getMaxValue(String unit)</td>
<td></td>
<td>Returns the property maximum value according to the specified unit.</td>
<td>63</td>
</tr>
<tr>
<td>Map getMetadata(String unit)</td>
<td></td>
<td>Returns metadata about the function property or operation parameter.</td>
<td>62</td>
</tr>
<tr>
<td>FunctionData getMinValue(String unit)</td>
<td></td>
<td>Returns the property minimum value according to the specified unit.</td>
<td>63</td>
</tr>
<tr>
<td>Object getResolution(String unit)</td>
<td></td>
<td>Returns the resolution value of specific range.</td>
<td>62</td>
</tr>
</tbody>
</table>
Field Detail

PROPERTY_ACCESS_READABLE
public static final int PROPERTY_ACCESS_READABLE = 1

Marks the readable function properties. The flag can be used as a part of bitmap value of PROPERTY_ACCESS. The readable access mandates function to provide a property getter method.

See Also: Function

PROPERTY_ACCESS_WRITABLE
public static final int PROPERTY_ACCESS_WRITABLE = 2

Marks the writable function properties. The flag can be used as a part of bitmap value of PROPERTY_ACCESS. The writable access mandates function to provide a property setter methods.

See Also: Function

PROPERTY_ACCESS_EVENTABLE
public static final int PROPERTY_ACCESS_EVENTABLE = 4

Marks the eventable function properties. The flag can be used as a part of bitmap value of PROPERTY_ACCESS.

See Also: Function

PROPERTY_ACCESS
public static final String PROPERTY_ACCESS = "property.access"

Metadata key, which value represents the access to the function property. The property value is a bitmap of Integer type. The bitmap can be any combination of:

- PROPERTY_ACCESS_READABLE
- PROPERTY_ACCESS_WRITABLE
- PROPERTY_ACCESS_EVENTABLE

For example, value Integer(3) means that the property is readable and writable, but not eventable.

The property access is available only for function properties and it's missing for the operation parameters.

DESCRIPTION
public static final String DESCRIPTION = "description"

Metadata key, which value represents the property description. The property value type is java.lang.String.

UNITS
public static final String UNITS = "units"
Metadata key, which value represents the property supported units. The property value type is `java.lang.String[]`. Each unit must follow those rules:

- The International System of Units must be used where it's applicable. For example, kg for kilogram and km for kilometre.
- If the unit name matches to an Unicode symbol name, the Unicode symbol must be used. For example, the degree unit matches to the Unicode degree sign (°).
- If the unit name doesn't match to an Unicode symbol, the unit symbol must be built by Unicode Basic Latin block of characters, superscript and subscript characters. For example, watt per square metre steradian is built by W/(m² sr), where ^2 is Unicode superscript two.

If those rules cannot be applied to the unit symbol, custom rules are allowed. A set of predefined unit symbols are available in the `Units` interface.

### Method Detail

#### `getMetadata` Method

```java
Map getMetadata(String unit)
```

Returns metadata about the function property or operation parameter. The keys of the `java.util.Map` result must be of `java.lang.String` type. Possible keys:

- **DESCRIPTION** - doesn't depend on the given unit.
- **PROPERTY_ACCESS** - available only for function property and missing for function operation parameters. It doesn't depend on the given unit.
- **UNITS** - doesn't depend on the given unit.
- custom key - can depend on the unit.

**Parameters:**
- `unit` - The unit to align the metadata if it's applicable. It can be null, which means that the default unit will be used.

**Returns:**
- The property metadata or `null` if no such metadata is available.

#### `getResolution` Method

```java
Object getResolution(String unit)
```

Throws `IllegalArgumentException` if the unit is not supported.

Returns the resolution value of specific range. For example, if the range is [0, 100], the resolution can be 10. That's the different between two values in series. The resolution type depends on the property type. If the property is using data bean like `LevelData`, the resolution will be a `BigDecimal`.

**Parameters:**
- `unit` - The unit to align the resolution, can be `null`.

**Returns:**
- The resolution according to the specified unit or `null` if no resolution is supported.

#### `getEnumValues` Method

```java
FunctionData[] getEnumValues(String unit)
```

Throws `IllegalArgumentException` if the unit is not supported.

Returns the property possible values according to the specified unit. If the unit is `null`, the values set is aligned to the default unit. If there is no such set of supported values, `null` is returned. The values must be sorted in increasing order.
Parameters:
  unit - The unit to align the supported values, can be null.

Returns:
The supported values according to the specified unit or null if no such values are supported. The values must be sorted in increasing order.

Throws:
IllegalArgumentException - If the unit is not supported.

getMinValue

FunctionData getMinValue(String unit)
  throws IllegalArgumentException

Returns the property minimum value according to the specified unit. If the unit is null, the minimum value is aligned to the default unit. If there is no minimum value, null is returned.

Parameters:
  unit - The unit to align the minimum value, can be null.

Returns:
The minimum value according to the specified unit or null if no minimum value is supported.

Throws:
IllegalArgumentException - If the unit is not supported.

getMaxValue

FunctionData getMaxValue(String unit)
  throws IllegalArgumentException

Returns the property maximum value according to the specified unit. If the unit is null, the maximum value is aligned to the default unit. If there is no maximum value, null is returned.

Parameters:
  unit - The unit to align the maximum value, can be null.

Returns:
The maximum value according to the specified unit or null if no maximum value is supported.

Throws:
IllegalArgumentException - If the unit is not supported.
Interface Units

public interface Units

Contains the most of the International System of Units unit symbols. The constant name represents the unit name. The constant value represents the unit symbol as it's defined in PropertyMetadata.UNITS.

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String AMPERE</td>
<td>69</td>
</tr>
<tr>
<td>Unit of electric current defined by the International System of Units (SI).</td>
<td></td>
</tr>
<tr>
<td>String AMPERE_PER_METRE</td>
<td>70</td>
</tr>
<tr>
<td>Unit of magnetic field strength.</td>
<td></td>
</tr>
<tr>
<td>String AMPERE_PER_SQUARE_METRE</td>
<td>70</td>
</tr>
<tr>
<td>Unit of current density.</td>
<td></td>
</tr>
<tr>
<td>String ANGSTROM</td>
<td>76</td>
</tr>
<tr>
<td>Unit of length.</td>
<td></td>
</tr>
<tr>
<td>String BAR</td>
<td>76</td>
</tr>
<tr>
<td>Unit of pressure.</td>
<td></td>
</tr>
<tr>
<td>String BARN</td>
<td>77</td>
</tr>
<tr>
<td>Unit of area.</td>
<td></td>
</tr>
<tr>
<td>String BECQUEREL</td>
<td>72</td>
</tr>
<tr>
<td>Unit of activity referred to a radionuclide.</td>
<td></td>
</tr>
<tr>
<td>String BEL</td>
<td>77</td>
</tr>
<tr>
<td>Unit of logarithmic ratio quantities.</td>
<td></td>
</tr>
<tr>
<td>String CANDELA</td>
<td>69</td>
</tr>
<tr>
<td>Unit of luminous intensity defined by the International System of Units (SI).</td>
<td></td>
</tr>
<tr>
<td>String CANDELA_PER_SQUARE_METRE</td>
<td>70</td>
</tr>
<tr>
<td>Unit of luminance.</td>
<td></td>
</tr>
<tr>
<td>String COULOMB</td>
<td>71</td>
</tr>
<tr>
<td>Unit of electronic charge, amount of electricity.</td>
<td></td>
</tr>
<tr>
<td>String COULOMB_PER_CUBIC_METRE</td>
<td>74</td>
</tr>
<tr>
<td>Unit of electric charge density.</td>
<td></td>
</tr>
<tr>
<td>String COULOMB_PER_KILOGRAM</td>
<td>75</td>
</tr>
<tr>
<td>Unit of exposure (x- and gamma-rays).</td>
<td></td>
</tr>
<tr>
<td>String COULOMB_PER_SQUARE_METRE</td>
<td>74</td>
</tr>
<tr>
<td>Unit of surface charge density, electric flux density, electric displacement.</td>
<td></td>
</tr>
<tr>
<td>String CUBIC_METRE</td>
<td>69</td>
</tr>
<tr>
<td>Unit of volume.</td>
<td></td>
</tr>
<tr>
<td>String CUBIC_METRE_PER_KILOGRAM</td>
<td>70</td>
</tr>
<tr>
<td>Unit of specific volume.</td>
<td></td>
</tr>
<tr>
<td>String DAY</td>
<td>76</td>
</tr>
<tr>
<td>Unit of time.</td>
<td></td>
</tr>
<tr>
<td>String DECIBEL</td>
<td>77</td>
</tr>
<tr>
<td>Unit of logarithmic ratio quantities.</td>
<td></td>
</tr>
<tr>
<td>String DEGREE</td>
<td>76</td>
</tr>
<tr>
<td>Unit of plane angle.</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DEGREE CELSIUS</td>
<td>Unit of Celsius temperature.</td>
</tr>
<tr>
<td>DYNE</td>
<td>Unit of force.</td>
</tr>
<tr>
<td>ERG</td>
<td>Unit of energy.</td>
</tr>
<tr>
<td>FARAD</td>
<td>Unit of capacitance.</td>
</tr>
<tr>
<td>FARAD PER METRE</td>
<td>Unit of permittivity.</td>
</tr>
<tr>
<td>GAL</td>
<td>Unit of acceleration.</td>
</tr>
<tr>
<td>GAUSS</td>
<td>Unit of magnetic flux density.</td>
</tr>
<tr>
<td>GRAY</td>
<td>Unit of absorbed dose, specific energy (imparted), kerma.</td>
</tr>
<tr>
<td>GRAY PER SECOND</td>
<td>Unit of absorbed dose rate.</td>
</tr>
<tr>
<td>HECTARE</td>
<td>Unit of area.</td>
</tr>
<tr>
<td>HENRY</td>
<td>Unit of inductance.</td>
</tr>
<tr>
<td>HENRY_PER METRE</td>
<td>Unit of permeability.</td>
</tr>
<tr>
<td>HERTZ</td>
<td>Unit of frequency.</td>
</tr>
<tr>
<td>HOUR</td>
<td>Unit of time.</td>
</tr>
<tr>
<td>JOULE</td>
<td>Unit of energy, work, amount of electricity.</td>
</tr>
<tr>
<td>JOULE PER CUBIC METRE</td>
<td>Unit of energy density.</td>
</tr>
<tr>
<td>JOULE PER KELVIN</td>
<td>Unit of heat capacity, entropy.</td>
</tr>
<tr>
<td>JOULE PER KILOGRAM</td>
<td>Unit of specific energy.</td>
</tr>
<tr>
<td>JOULE PER KILOGRAM KELVIN</td>
<td>Unit of specific heat capacity, specific entropy.</td>
</tr>
<tr>
<td>JOULE_PER_MOLE</td>
<td>Unit of molar energy.</td>
</tr>
<tr>
<td>JOULE_PER_MOLE_KELVIN</td>
<td>Unit of molar entropy, molar heat capacity.</td>
</tr>
<tr>
<td>KATAL</td>
<td>Unit of catalytic activity.</td>
</tr>
<tr>
<td>KATAL PER CUBIC METRE</td>
<td>Unit of catalytic activity concentration.</td>
</tr>
<tr>
<td>KELVIN</td>
<td>Unit of thermodynamic temperature defined by the International System of Units (SI).</td>
</tr>
<tr>
<td>KILOGRAM</td>
<td>Unit of mass defined by the International System of Units (SI).</td>
</tr>
<tr>
<td>String</td>
<td>Unit</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>KILOGRAM_PER_CUBIC_METRE</td>
<td>Unit of density, mass density, mass concentration.</td>
</tr>
<tr>
<td>KILOGRAM_PER_SQUARE_METRE</td>
<td>Unit of surface density.</td>
</tr>
<tr>
<td>KNOT</td>
<td>Unit of speed.</td>
</tr>
<tr>
<td>LITRE</td>
<td>Unit of volume.</td>
</tr>
<tr>
<td>LUMEN</td>
<td>Unit of luminous flux.</td>
</tr>
<tr>
<td>LUX</td>
<td>Unit of illuminance.</td>
</tr>
<tr>
<td>MAXWELL</td>
<td>Unit of magnetic flux.</td>
</tr>
<tr>
<td>METRE</td>
<td>Unit of length defined by the International System of Units (SI).</td>
</tr>
<tr>
<td>METRE_PER_SECOND</td>
<td>Unit of speed, velocity.</td>
</tr>
<tr>
<td>METRE_PER_SECOND_SQUARED</td>
<td>Unit of acceleration.</td>
</tr>
<tr>
<td>MILLIMETRE_OF_MERCURY</td>
<td>Unit of pressure.</td>
</tr>
<tr>
<td>MOLE</td>
<td>Unit of amount of substance defined by the International System of Units (SI).</td>
</tr>
<tr>
<td>MOLE_PER_CUBIC_METRE</td>
<td>Unit of amount concentration, concentration.</td>
</tr>
<tr>
<td>NAUTICAL_MILE</td>
<td>Unit of distance.</td>
</tr>
<tr>
<td>NEPER</td>
<td>Unit of logarithmic ratio quantities.</td>
</tr>
<tr>
<td>NEWTON</td>
<td>Unit of force.</td>
</tr>
<tr>
<td>NEWTON_METRE</td>
<td>Unit of moment of force.</td>
</tr>
<tr>
<td>NEWTON_PER_METRE</td>
<td>Unit of surface tension.</td>
</tr>
<tr>
<td>OERSTED</td>
<td>Unit of magnetic field.</td>
</tr>
<tr>
<td>OHM</td>
<td>Unit of electric resistance.</td>
</tr>
<tr>
<td>PASCAL</td>
<td>Unit of pressure, stress.</td>
</tr>
<tr>
<td>PASCAL_SECOND</td>
<td>Unit of dynamic viscosity.</td>
</tr>
<tr>
<td>PHOT</td>
<td>Unit of illuminance.</td>
</tr>
<tr>
<td>PLANE_ANGLE_MINUTE</td>
<td>Unit of plane angle.</td>
</tr>
<tr>
<td>PLANE_ANGLE_SECOND</td>
<td>Unit of plane angle.</td>
</tr>
<tr>
<td>String</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>POISE</td>
<td>Unit of dynamic viscosity.</td>
</tr>
<tr>
<td>PREFIX ATTO</td>
<td>Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.</td>
</tr>
<tr>
<td>PREFIX CENTI</td>
<td>Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.</td>
</tr>
<tr>
<td>PREFIX DECA</td>
<td>Adopted prefix symbol to form the symbols of the decimal multiples of SI units.</td>
</tr>
<tr>
<td>PREFIX DECI</td>
<td>Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.</td>
</tr>
<tr>
<td>PREFIX EXA</td>
<td>Adopted prefix symbol to form the symbols of the decimal multiples of SI units.</td>
</tr>
<tr>
<td>PREFIX FEMTO</td>
<td>Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.</td>
</tr>
<tr>
<td>PREFIX GIGA</td>
<td>Adopted prefix symbol to form the symbols of the decimal multiples of SI units.</td>
</tr>
<tr>
<td>PREFIX HECTO</td>
<td>Adopted prefix symbol to form the symbols of the decimal multiples of SI units.</td>
</tr>
<tr>
<td>PREFIX KILO</td>
<td>Adopted prefix symbol to form the symbols of the decimal multiples of SI units.</td>
</tr>
<tr>
<td>PREFIX MEGA</td>
<td>Adopted prefix symbol to form the symbols of the decimal multiples of SI units.</td>
</tr>
<tr>
<td>PREFIX MICRO</td>
<td>Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.</td>
</tr>
<tr>
<td>PREFIX MILLI</td>
<td>Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.</td>
</tr>
<tr>
<td>PREFIX NANO</td>
<td>Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.</td>
</tr>
<tr>
<td>PREFIX PICO</td>
<td>Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.</td>
</tr>
<tr>
<td>PREFIX YOCTO</td>
<td>Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.</td>
</tr>
<tr>
<td>PREFIX YOTTA</td>
<td>Adopted prefix symbol to form the symbols of the decimal multiples of SI units.</td>
</tr>
<tr>
<td>PREFIX ZEPTO</td>
<td>Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.</td>
</tr>
<tr>
<td>PREFIX ZETTA</td>
<td>Adopted prefix symbol to form the symbols of the decimal multiples of SI units.</td>
</tr>
<tr>
<td>RADIAN</td>
<td>Unit of plane angle.</td>
</tr>
<tr>
<td>RADIAN_PER_SECOND</td>
<td>Unit of angular velocity.</td>
</tr>
<tr>
<td>RADIAN_PER_SECOND_SQUARED</td>
<td>Unit of angular acceleration.</td>
</tr>
<tr>
<td>RECIPROCAL_Metre</td>
<td>Unit of wavenumber.</td>
</tr>
<tr>
<td>SECOND</td>
<td>Unit of time defined by the International System of Units (SI).</td>
</tr>
<tr>
<td>SIEMENS</td>
<td>Unit of electric conductance.</td>
</tr>
</tbody>
</table>
### Interface Units

<table>
<thead>
<tr>
<th>String</th>
<th>Unit Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIEVERT</td>
<td>Unit of dose equivalent, ambient dose equivalent, directional dose equivalent, personal dose equivalent.</td>
<td>73</td>
</tr>
<tr>
<td>SQUARE METRE</td>
<td>Unit of area.</td>
<td>69</td>
</tr>
<tr>
<td>STERADIAN</td>
<td>Unit of solid angle.</td>
<td>70</td>
</tr>
<tr>
<td>STILB</td>
<td>Unit of luminance.</td>
<td>78</td>
</tr>
<tr>
<td>STOKES</td>
<td>Unit of kinematic viscosity.</td>
<td>77</td>
</tr>
<tr>
<td>TESLA</td>
<td>Unit of magnetic flux density.</td>
<td>72</td>
</tr>
<tr>
<td>TIME MINUTE</td>
<td>Unit of time.</td>
<td>75</td>
</tr>
<tr>
<td>TONNE</td>
<td>Unit of mass.</td>
<td>76</td>
</tr>
<tr>
<td>VOLT</td>
<td>Unit of electric potential difference, electromotive force.</td>
<td>71</td>
</tr>
<tr>
<td>VOLT PER METRE</td>
<td>Unit of electric field strength.</td>
<td>74</td>
</tr>
<tr>
<td>WATT</td>
<td>Unit of power, radiant flux.</td>
<td>71</td>
</tr>
<tr>
<td>WATT PER METRE KELVIN</td>
<td>Unit of thermal conductivity.</td>
<td>74</td>
</tr>
<tr>
<td>WATT PER SQUARE METRE</td>
<td>Unit of heat flux density, irradiance.</td>
<td>73</td>
</tr>
<tr>
<td>WATT PER SQUARE METRE STERADIAN</td>
<td>Unit of radiance.</td>
<td>75</td>
</tr>
<tr>
<td>WATT PER STERADIAN</td>
<td>Unit of radiant intensity.</td>
<td>75</td>
</tr>
<tr>
<td>WEBER</td>
<td>Unit of magnetic flux.</td>
<td>72</td>
</tr>
</tbody>
</table>

### Field Detail

**METRE**

```java
public static final String METRE = "m"
```

Unit of length defined by the International System of Units (SI). It's one of be base units called metre.

**KILOGRAM**

```java
public static final String KILOGRAM = "kg"
```

Unit of mass defined by the International System of Units (SI). It's one of be base units called kilogram.

**SECOND**

```java
public static final String SECOND = "s"
```

Unit of time defined by the International System of Units (SI). It's one of be base units called second.
**Interface Units**

**AMPERE**

public static final String AMPERE = "A"

Unit of electric current defined by the International System of Units (SI). It's one of the base units called ampere.

**KELVIN**

public static final String KELVIN = "\u212a"

Unit of thermodynamic temperature defined by the International System of Units (SI). It's one of the base units called kelvin.

**MOLE**

public static final String MOLE = "mol"

Unit of amount of substance defined by the International System of Units (SI). It's one of the base units called mole.

**CANDELA**

public static final String CANDELA = "cd"

Unit of luminous intensity defined by the International System of Units (SI). It's one of the base units called candela.

**SQUARE_METRE**

public static final String SQUARE_METRE = "m\u00b2"

Unit of area. It's one of the coherent derived units in the SI expressed in terms of base units. The unit is called square metre.

**CUBIC_METRE**

public static final String CUBIC_METRE = "m\u00b3"

Unit of volume. It's one of the coherent derived units in the SI expressed in terms of base units. The unit is called cubic metre.

**METRE_PER_SECOND**

public static final String METRE_PER_SECOND = "m/s"

Unit of speed, velocity. It's one of the coherent derived units in the SI expressed in terms of base units. The unit is called metre per second.

**METRE_PER_SECOND_SQUARED**

public static final String METRE_PER_SECOND_SQUARED = "m/s\u00b2"

Unit of acceleration. It's one of the coherent derived units in the SI expressed in terms of base units. The unit is called metre per second squared.

**RECIPROCAL_METRE**

public static final String RECIPROCAL_METRE = "m\u207b\u00b9"

Unit of wavenumber. It's one of the coherent derived units in the SI expressed in terms of base units. The unit is called reciprocal metre.
**Interface Units**

**KILOGRAM_PER_CUBIC_METRE**

```java
public static final String KILOGRAM_PER_CUBIC_METRE = "kg/m\u00b3"
```

Unit of density, mass density, mass concentration. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called kilogram per cubic metre.

**KILOGRAM_PER_SQUARE_METRE**

```java
public static final String KILOGRAM_PER_SQUARE_METRE = "kg/m\u00b2"
```

Unit of surface density. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called kilogram per square metre.

**CUBIC_METRE_PER_KILOGRAM**

```java
public static final String CUBIC_METRE_PER_KILOGRAM = "m\u00b3/kg"
```

Unit of specific volume. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called cubic metre per kilogram.

**AMPERE_PER_SQUARE_METRE**

```java
public static final String AMPERE_PER_SQUARE_METRE = "A/m\u00b2"
```

Unit of current density. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called ampere per square metre.

**AMPERE_PER_METRE**

```java
public static final String AMPERE_PER_METRE = "A/m"
```

Unit of magnetic field strength. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called ampere per metre.

**MOLE_PER_CUBIC_METRE**

```java
public static final String MOLE_PER_CUBIC_METRE = "mol/m\u00b3"
```

Unit of amount concentration, concentration. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called mole per cubic metre.

**CANDELA_PER_SQUARE_METRE**

```java
public static final String CANDELA_PER_SQUARE_METRE = "cd/m\u00b2"
```

Unit of luminance. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called candela per square metre.

**RADIAN**

```java
public static final String RADIAN = "rad"
```

Unit of plane angle. It's one of the coherent derived units in the SI with special names and symbols. The unit is called radian.

**STERADIAN**

```java
public static final String STERADIAN = "sr"
```

Unit of solid angle. It's one of the coherent derived units in the SI with special names and symbols. The unit is called steradian.
Interface Units

**HERTZ**

public static final String Hertz = "Hz"

Unit of frequency. It's one of the coherent derived units in the SI with special names and symbols. The unit is called hertz.

**NEWTON**

public static final String Newton = "N"

Unit of force. It's one of the coherent derived units in the SI with special names and symbols. The unit is called newton.

**PASCAL**

public static final String Pascal = "Pa"

Unit of pressure, stress. It's one of the coherent derived units in the SI with special names and symbols. The unit is called pascal.

**JOULE**

public static final String Joule = "J"

Unit of energy, work, amount of electricity. It's one of the coherent derived units in the SI with special names and symbols. The unit is called joule.

**WATT**

public static final String Watt = "W"

Unit of power, radiant flux. It's one of the coherent derived units in the SI with special names and symbols. The unit is called watt.

**COULOMB**

public static final String Coulomb = "C"

Unit of electronic charge, amount of electricity. It's one of the coherent derived units in the SI with special names and symbols. The unit is called coulomb.

**VOLT**

public static final String Volt = "V"

Unit of electric potential difference, electromotive force. It's one of the coherent derived units in the SI with special names and symbols. The unit is called volt.

**FARAD**

public static final String Farad = "F"

Unit of capacitance. It's one of the coherent derived units in the SI with special names and symbols. The unit is called farad.

**OHM**

public static final String Ohm = "Ω"
**Interface Units**

**SIEMENS**

```java
public static final String SIEMENS = "S"
```

Unit of electric conductance. It's one of the coherent derived units in the SI with special names and symbols. The unit is called siemens.

**WEBER**

```java
public static final String WEBER = "Wb"
```

Unit of magnetic flux. It's one of the coherent derived units in the SI with special names and symbols. The unit is called weber.

**TESLA**

```java
public static final String TESLA = "T"
```

Unit of magnetic flux density. It's one of the coherent derived units in the SI with special names and symbols. The unit is called tesla.

**HENRY**

```java
public static final String HENRY = "H"
```

Unit of inductance. It's one of the coherent derived units in the SI with special names and symbols. The unit is called henry.

**DEGREE_CELSIUS**

```java
public static final String DEGREE_CELSIUS = "\u2103"
```

Unit of Celsius temperature. It's one of the coherent derived units in the SI with special names and symbols. The unit is called degree Celsius.

**LUMEN**

```java
public static final String LUMEN = "lm"
```

Unit of luminous flux. It's one of the coherent derived units in the SI with special names and symbols. The unit is called lumen.

**LUX**

```java
public static final String LUX = "lx"
```

Unit of illuminance. It's one of the coherent derived units in the SI with special names and symbols. The unit is called lux.

**BECQUEREL**

```java
public static final String BECQUEREL = "Bq"
```

Unit of activity referred to a radionuclide. It's one of the coherent derived units in the SI with special names and symbols. The unit is called becquerel.

**GRAY**

```java
public static final String GRAY = "Gy"
```

Unit of absorbed dose, specific energy (imparted), kerma. It's one of the coherent derived units in the SI with special names and symbols. The unit is called gray.
**Interface Units**

**SIEVERT**

public static final String SIEVERT = "Sv"

Unit of dose equivalent, ambient dose equivalent, directional dose equivalent, personal dose equivalent. It's one of the coherent derived units in the SI with special names and symbols. The unit is called sievert.

**KATAL**

public static final String KATAL = "kat"

Unit of catalytic activity. It's one of the coherent derived units in the SI with special names and symbols. The unit is called katal.

**PASCAL_SECOND**

public static final String PASCAL_SECOND = "Pa s"

Unit of dynamic viscosity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called pascal second.

**NEWTON_METRE**

public static final String NEWTON_METRE = "N m"

Unit of moment of force. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called newton metre.

**NEWTON_PER_METRE**

public static final String NEWTON_PER_METRE = "N/m"

Unit of surface tension. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called newton per metre.

**RADIAN_PER_SECOND**

public static final String RADIAN_PER_SECOND = "rad/s"

Unit of angular velocity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called radian per second.

**RADIAN_PER_SECOND_SQUARED**

public static final String RADIAN_PER_SECOND_SQUARED = "rad/s\u00b2"

Unit of angular acceleration. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called radian per second squared.

**WATT_PER_SQUARE_METRE**

public static final String WATT_PER_SQUARE_METRE = "W/m\u00b2"

Unit of heat flux density, irradiance. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called watt per square metre.

**JOULE_PER_KELVIN**

public static final String JOULE_PER_KELVIN = "J/K"

Unit of heat capacity, entropy. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per kelvin.
Interface Units

**JOULE_PER_KILOGRAM_KELVIN**

public static final String JOULE_PER_KILOGRAM_KELVIN = "J/(kg K)"

Unit of specific heat capacity, specific entropy. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per kilogram kelvin.

**JOULE_PER_KILOGRAM**

public static final String JOULE_PER_KILOGRAM = "J/kg"

Unit of specific energy. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per kilogram.

**WATT_PER_METRE_KELVIN**

public static final String WATT_PER_METRE_KELVIN = "W/(m K)"

Unit of thermal conductivity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called watt per metre kelvin.

**JOULE_PER_CUBIC_METRE**

public static final String JOULE_PER_CUBIC_METRE = "J/m³"

Unit of energy density. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per cubic metre.

**VOLT_PER_METRE**

public static final String VOLT_PER_METRE = "V/m"

Unit of electric field strength. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called volt per metre.

**COULOMB_PER_CUBIC_METRE**

public static final String COULOMB_PER_CUBIC_METRE = "C/m³"

Unit of electric charge density. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called coulomb per cubic metre.

**COULOMB_PER_SQUARE_METRE**

public static final String COULOMB_PER_SQUARE_METRE = "C/m²"

Unit of surface charge density, electric flux density, electric displacement. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called coulomb per square metre.

**FARAD_PER_METRE**

public static final String FARAD_PER_METRE = "F/m"

Unit of permittivity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called farad per metre.

**HENRY_PER_METRE**

public static final String HENRY_PER_METRE = "H/m"

Unit of permeability. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called henry per metre.
Interface Units

**JOULE_PER_MOLE**

public static final String JOULE_PER_MOLE = "J/mol"

Unit of molar energy. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per mole.

**JOULE_PER_MOLE_KELVIN**

public static final String JOULE_PER_MOLE_KELVIN = "J/(mol K)"

Unit of molar entropy, molar heat capacity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per mole kelvin.

**COULOMB_PER_KILOGRAM**

public static final String COULOMB_PER_KILOGRAM = "C/kg"

Unit of exposure (x- and gamma-rays). It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called coulomb per kilogram.

**GRAY_PER_SECOND**

public static final String GRAY_PER_SECOND = "Gy/s"

Unit of absorbed dose rate. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called gray per second.

**WATT_PER_STERADIAN**

public static final String WATT_PER_STERADIAN = "W/sr"

Unit of radiant intensity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called watt per steradian.

**WATT_PER_SQUARE_METRE_STERADIAN**

public static final String WATT_PER_SQUARE_METRE_STERADIAN = "W/(m\(^2\) sr)"

Unit of radiance. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called watt per square metre steradian.

**KATAL_PER_CUBIC_METRE**

public static final String KATAL_PER_CUBIC_METRE = "kat/m\(^3\)"

Unit of catalytic activity concentration. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called katal per cubic metre.

**TIME_MINUTE**

public static final String TIME_MINUTE = "min"

Unit of time. It's one of non-SI units accepted for use with the International System of Units. The unit is called minute.

**HOUR**

public static final String HOUR = "h"

Unit of time. It's one of non-SI units accepted for use with the International System of Units. The unit is called hour.
Interface Units

**DAY**

```java
public static final String DAY = "d"
```

Unit of time. It's one of non-SI units accepted for use with the International System of Units. The unit is called day.

**DEGREE**

```java
public static final String DEGREE = "\u00b0"
```

Unit of plane angle. It's one of non-SI units accepted for use with the International System of Units. The unit is called degree.

**PLANE_ANGLE_MINUTE**

```java
public static final String PLANE_ANGLE_MINUTE = "\u2032"
```

Unit of plane angle. It's one of non-SI units accepted for use with the International System of Units. The unit is called minute.

**PLANE_ANGLE_SECOND**

```java
public static final String PLANE_ANGLE_SECOND = "\u2033"
```

Unit of plane angle. It's one of non-SI units accepted for use with the International System of Units. The unit is called second.

**HECTARE**

```java
public static final String HECTARE = "ha"
```

Unit of area. It's one of non-SI units accepted for use with the International System of Units. The unit is called hectare.

**LITRE**

```java
public static final String LITRE = "l"
```

Unit of volume. It's one of non-SI units accepted for use with the International System of Units. The unit is called litre. International System of Units accepts two symbols: lower-case l and capital L. That constant value is using the lower-case l.

**TONNE**

```java
public static final String TONNE = "t"
```

Unit of mass. It's one of non-SI units accepted for use with the International System of Units. The unit is called tonne.

**BAR**

```java
public static final String BAR = "bar"
```

Unit of pressure. It's one of other non-SI units. The unit is called bar.

**MILLIMETRE_OF_MERCURY**

```java
public static final String MILLIMETRE_OF_MERCURY = "mmHg"
```

Unit of pressure. It's one of other non-SI units. The unit is called millimetre of mercury.

**ANGSTROM**

```java
public static final String ANGSTROM = "\u212b"
```
Interface Units

Unit of length. It's one of other non-SI units. The unit is called angstrom.

NAUTICAL_MILE

public static final String NAUTICAL_MILE = "M"

Unit of distance. It's one of other non-SI units. The unit is called nautical mile.

BARN

public static final String BARN = "b"

Unit of area. It's one of other non-SI units. The unit is called barn.

KNOT

public static final String KNOT = "kn"

Unit of speed. It's one of other non-SI units. The unit is called knot.

NEPER

public static final String NEPER = "Np"

Unit of logarithmic ratio quantities. It's one of other non-SI units. The unit is called neper.

BEL

public static final String BEL = "B"

Unit of logarithmic ratio quantities. It's one of other non-SI units. The unit is called bel.

DECIBEL

public static final String DECIBEL = "dB"

Unit of logarithmic ratio quantities. It's one of other non-SI units. The unit is called decibel.

ERG

public static final String ERG = "erg"

Unit of energy. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called erg.

DYNE

public static final String DYNE = "dyn"

Unit of force. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called dyne.

POISE

public static final String POISE = "p"

Unit of dynamic viscosity. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called poise.

STOKES

public static final String STOKES = "St"

Unit of kinematic viscosity. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called stokes.
**STILB**

public static final String STILB = "sb"

Unit of luminance. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called stilb.

**PHOT**

public static final String PHOT = "ph"

Unit of illuminance. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called phot.

**GAL**

public static final String GAL = "Gal"

Unit of acceleration. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called gal.

**MAXWELL**

public static final String MAXWELL = "Mx"

Unit of magnetic flux. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called maxwell.

**GAUSS**

public static final String GAUSS = "G"

Unit of magnetic flux density. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called gauss.

**OERSTED**

public static final String OERSTED = "Oe"

Unit of magnetic field. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called oersted.

**PREFIX_DECA**

public static final String PREFIX_DECA = "da"

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called deca and represents the 1st power of ten.

**PREFIX_HECTO**

public static final String PREFIX_HECTO = "h"

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called hecto and represents the 2nd power of ten.

**PREFIX_KILO**

public static final String PREFIX_KILO = "k"

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called kilo and represents the 3rd power of ten.
Interface Units

**PREFIX_MEGA**

```java
public static final String PREFIX_MEGA = "M"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called mega and represents the 6th power of ten.

**PREFIX_GIGA**

```java
public static final String PREFIX_GIGA = "G"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called giga and represents the 9th power of ten.

**PREFIX_EXA**

```java
public static final String PREFIX_EXA = "E"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called exa and represents the 18th power of ten.

**PREFIX_ZETTA**

```java
public static final String PREFIX_ZETTA = "Z"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called zetta and represents the 21th power of ten.

**PREFIX_YOTTA**

```java
public static final String PREFIX_YOTTA = "Y"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called yotta and represents the 24th power of ten.

**PREFIX_DECI**

```java
public static final String PREFIX_DECI = "d"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called deci and represents the 1st negative power of ten.

**PREFIX_CENTI**

```java
public static final String PREFIX_CENTI = "c"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called centi and represents the 2nd negative power of ten.

**PREFIX_MILLI**

```java
public static final String PREFIX_MILLI = "m"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called milli and represents the 3rd negative power of ten.

**PREFIX_MICRO**

```java
public static final String PREFIX_MICRO = "\u00b5"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called micro and represents the 6th negative power of ten.
Interface Units

**PREFIX_NANO**

```java
public static final String PREFIX_NANO = "n"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called nano and represents the 9th negative power of ten.

**PREFIX_PICO**

```java
public static final String PREFIX_PICO = "p"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called pico and represents the 12th negative power of ten.

**PREFIX_FEMTO**

```java
public static final String PREFIX_FEMTO = "f"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called femto and represents the 15th negative power of ten.

**PREFIX_ATTO**

```java
public static final String PREFIX_ATTO = "a"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called atto and represents the 18th negative power of ten.

**PREFIX_ZEPTO**

```java
public static final String PREFIX_ZEPTO = "z"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called zepto and represents the 21st negative power of ten.

**PREFIX_YOCTO**

```java
public static final String PREFIX_YOCTO = "y"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called yocto and represents the 24th negative power of ten.

Java API documentation generated with [DocFlex/Doclet](https://docflex.com) v1.5.6

DocFlex/Doclet is both a multi-format Javadoc doclet and a free edition of [DocFlex/Javadoc](https://docflex.com). If you need to customize your Javadoc without writing a full-blown doclet from scratch, DocFlex/Javadoc may be the only tool able to help you! Find out more at [www.docflex.com](http://www.docflex.com).
8 Considered Alternatives

8.1 Use Configuration Admin to update the Device service properties

OSGi service properties are used to represent the Device service properties. The properties can be updated with the help of `org.osgi.framework.ServiceRegistration.setProperties(Dictionary)` method. The service registration is intended for a private usage and should not be shared between the bundles.

The current design provides set methods, which can be used when an external application wants to modify the Device service properties. It's simple and a part of Device interface. We have to define a new permission check, because there is no such protection to `org.osgi.framework.ServiceRegistration.setProperties` method.

Considered alternative was about property update based on configuration update in the Configuration Admin service. The Device service properties can be updated when the corresponding configuration properties are updated. The disadvantages here are:

- Device properties duplication – they are stored in the device configuration and in the Device service properties.
- Possible performance issue when a lot of devices are used.

8.2 DeviceAdmin interface availability

DeviceAdmin service was removed from the current RFC document. That management functionality can be provided by a different specification document. That considered alternative is kept for completeness.

DeviceAdmin service can simplify the device service registration. It hides the implementation details i.e. realize program to an interface rather than to an implementation.

The considered alternative is not to use that interface and to register the Device service implementation to the OSGi service registry. Here are two code snippets, which demonstrates positives and negatives:

1. Without DeviceAdmin
   ```java
   Map ipCameraProps = new HashMap(3, 1F);
   ipCameraProps.put("IP.Camera.Address", "192.168.0.21");
   ipCameraProps.put("IP.Camera.Username", "test");
   ipCameraProps.put("IP.Camera.Password", "test");
   //WARNING - an access to implementation class, which should be bundle private
   IPCameraDeviceImpl ipCameraImpl = new IPCameraDeviceImpl(ipCameraProps);
   ipCameraImpl.register(bundleContext);
   // play the video stream...
   ```
// remove the device
ipCameraImpl.unregister();

That snippet demonstrate program to implementation rather than an interface, which break basic OOP rule.

2. With DeviceAdmin

Map ipCameraProps = new HashMap(3, 1F);
ipCameraProps.put("IP.Camera.Address", "192.168.0.21");
ipCameraProps.put("IP.Camera.Username", "test");
ipCameraProps.put("IP.camera.Password", "test");

DeviceAdmin ipCameraDeviceAdmin = getIPCameraDeviceAdmin();
Device ipCamera = ipCameraDeviceAdmin.add(ipCameraProps);

// play the device video stream
// remove the device
ipCamera.remove();

It demonstrate program to interface rather than an implementation, which is the correct approach.

8.3 Access helper methods removal of FunctionalDevice

org.osgi.service.functionaldevice.FunctionalDevice.getChildren(),
or.org.osgi.service.functionaldevice.FunctionalDevice.getParent() and
org.osgi.service.functionaldevice.FunctionalDevice.getReferences() were removed, because they provided
access to the FunctionalDevice services outside the OSGi service registry. It can be problematic in various
scenarios like:

- The service Find Hook can be ignored.
- No service unget is possible for such shared service instances.
- The dependency tools based on the service registry cannot track such sharings.

9 Security Considerations

9.1 Device Permission

The device permission controls the bundle's authority to perform specific privileged administrative operations on
the devices. The action for this permission is:

<table>
<thead>
<tr>
<th>Action</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION_REMOVE</td>
<td>Device.remove()</td>
</tr>
</tbody>
</table>
The name of the permission is a filter based. For more details about filter based permissions, see OSGi Core Specification, Filter Based Permissions. The filter provides an access to all device service properties. The service property names are case insensitive. The filter attribute names are processed in a case insensitive manner. For example, the operator can give a bundle the permission to only manage devices of vendor “acme”:

```java
org.osgi.service.dal.DevicePermission("dal.device.hardware.vendor=acme", ...)
```

The permission action allows the operator to assign only the necessary permissions to the bundle. For example, the management bundle can have permission to remove all registered devices:

```java
org.osgi.service.dal.DevicePermission("*", "remove")
```

The code that needs to check the device permission must always use the constructor that takes the device as a parameter `DevicePermission(Device, String)` with a single action. For example, the implementation of `org.osgi.service.dal.Device.remove()` method must check that the caller has an access to the operation:

```java
public class DeviceImpl implements Device {
    public void start() {
        securityManager.checkPermission(new DevicePermission(this, "remove");
    }
}
```

### 9.2 Required Permissions

The Functional Device implementation must check the caller for the appropriate Functional Device Permission before execution of the real operation actions like remove. Once the Functional Device Permission is checked against the caller the implementation will proceed with the actual operation. The operation can require a number of other permissions to complete. The implementation must isolate the caller from such permission checks by use of proper privileged blocks.

### 10 Document Support

#### 10.1 References

4. Unicode Standard Annex #15, Unicode Normalization Forms

#### 10.2 Author’s Address
10.3 Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Abstraction Layer</td>
<td>Unifies the work with devices provided by different protocols.</td>
</tr>
<tr>
<td>Device Abstraction API</td>
<td>Unified API for management of devices provided by different protocols.</td>
</tr>
<tr>
<td>Device Abstraction Adapter</td>
<td>Examples for such adapters are ZigBee Adapter, Z-Wave Adapter etc. Provides support for a particular device protocol to Device Abstraction Layer. The adapter integrates the protocol specific driver devices.</td>
</tr>
</tbody>
</table>

10.4 End of Document
Abstract
This specification defines the Java API to discover and control EnOcean devices on the OSGi platform and according to OSGi service design patterns. This API maps the representation model of EnOcean entities defined by EnOcean Equipment Profiles standard into Java classes. OSGi service design patterns are used on the one hand for dynamic discovery, control and eventing of local and networked devices and on the other hand for dynamic network advertising and control of local OSGi services.
0 Document Information

License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such as 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.

Trademarks

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

Feedback

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

Table of Contents

0 Document Information.................................................................................................................2
  License..................................................................................................................................... 2
  Trademarks............................................................................................................................... 3
  Feedback................................................................................................................................... 3
  Table of Contents...................................................................................................................... 3
  Terminology and Document Conventions.............................................................................. 4
  Revision History....................................................................................................................... 4
1 Introduction................................................................................................................................ 11
2 Application Domain.................................................................................................................. 11
  System Architecture................................................................................................................ 11
  EnOcean Stack.......................................................................................................................... 12
  EnOcean Equipment Profiles (EEP)...................................................................................... 13
3 Problem Description.................................................................................................................. 14
4 Requirements........................................................................................................................... 14
5 Technical Solution.................................................................................................................... 16
6 Initial Spec Chapter.................................................................................................................. 17
  Introduction.............................................................................................................................. 17
  Essentials................................................................................................................................. 17
  Entities...................................................................................................................................... 18
  Operation Summary................................................................................................................ 20
  EnOcean Base Driver.............................................................................................................. 21
Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].

Source code is shown in this typeface.

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>
</table>
| Initial  | February, 26th, 2013 | Maïlys Robin, France Telecom Orange, mrobin.ext@orange.com  
|          |                    | Victor Perron, France Telecom Orange, victor.perron@orange.fr           |
| First draft | April 4th, 2013   | A. Bottaro, France Telecom Orange, M. Robin, France Telecom Orange,  
|          |                    | V. Perron, France Telecom Orange                                         |
| Revision 1 | April 18th, 2013  | V. Perron, France Telecom Orange  
|          |                    | • Rename EO* concepts into EnOcean*  
|          |                    | • Use a .learnedDevices property instead of getLearnedDevices()  
|          |                    | • Add a link between “Client Bundle” and “EnOceanDevice”              |
| Revision 2 | May 14th, 2013    | V. Perron, France Telecom Orange  
|          |                    | A. Bottaro, France Telecom Orange  
|          |                    | • Addition of the EnOceanMessage, EnOceanChannel, EnOceanScaledChannel,  
|          |                    | EnOceanEnumChannel, EnOceanEnumChannelRange interfaces  
|          |                    | • Rewrite of the EnOceanProfile, EnOceanRPC, EnOceanRMCC specification  
|          |                    | • Revision of the main and EnOceanDevice diagrams  
|          |                    | • Addition of the known EnOcean Exceptions  
|          |                    | • Addition of the EnOcean Event API section  
|          |                    | • EnOcean networking explanations                                         |
| Revision 3 | May 20th, 2013    | V. Perron, France Telecom Orange  
|          |                    | A. Bottaro, France Telecom Orange  
|          |                    | N. Portinaro, Telecom Italia  
|          |                    | A. Kraft, Deutsche Telekom  
|          |                    | • Take N. Portinaro's and A. Kraft’s remarks about send() standardization and  
|          |                    | level of detail  
|          |                    | • Remove EnOceanProfile notion in profit of EnOceanMessage.  
|          |                    | • Merged together RPC and RMCC notions.  
|          |                    | • The heavy changes to EnOceanMessage and EnOceanChannel types  
|          |                    | Introduced the EnOceanChannelDescription type that follows a more common  
|          |                    | design with UPnP and ZigBee device services.                             |
| Revision 4 | May 27th, 2013    | V. Perron, France Telecom Orange  
|          |                    | A. Bottaro, France Telecom Orange  
|          |                    | • Add support for Security  
|          |                    | • Challenge Generic Profiles support  
|          |                    | • Convergence towards EnOcean Link notions of Channels  
|          |                    | • Improve EnOceanHost notion into EnOceanGatewayChip  
<p>|          |                    | • EnOceanDevice EXPORT situation should work.                           |</p>
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Revision 5 | June 2nd, 2013 | V. Perron, France Telecom Orange  
A. Bottaro, France Telecom Orange  
N. Portinaro, Telecom Italia  
- Discussion is ongoing within OSGi members about the use of EnOceanHost as a low-level notion or bundled inside of the base driver; for now, only chip configuration is available, not send methods.  
- Use protected getters and setters instead of plain properties for security objects.  
- Add the repeater notion to EnOcean device.  
- Move any non-filtering property to a method form.  
- Discussed setChannels() and getChannels() methods that would allow for a generic implementation of a Message, finally not integrated. |
| Revision 6 | June 8th, 2013 | V. Perron, France Telecom Orange  
A. Bottaro, France Telecom Orange  
- Remove the Repeater notion from EnOcean Device and keep it only at the EnOceanHost level.  
- Add the sendSecureTeachIn() method  
- Some overall cleanup; question to reintegrate SmartAck. |
| Revision 7 | June 19th, 2013 | V. Perron, France Telecom Orange  
A. Bottaro, France Telecom Orange  
N. Portinaro, Telecom Italia  
- Overall cleanups: the RFC’s style has been rewritten in order to have less inclusions of Java-like text and be more descriptive. The Java specification, generated from the Javadoc, has been move to the end of the document, as what has been done with other service specifications.  
- EnOceanDevice: sendTeachIn, sendSecureTeachIn are removed in favor of a send(TeachInMessage). Provide setters for the dynamic, implementation-independant properties, like the senderId, security features, etc.  
- EnOceanMessage: the STATUS field is no more a filtering property, it carries not enough information and changes too often to be used as such. A getSubMessageCount() method has been added to help serializers in the case of multiple-frame messages. Those should be supported by the implementation transparently.  
- EnOceanChannel: Add the rawValue property that stores the value of the channel in bytes. Add setRawValue() and setValue() methods to enable for dynamic rewrite of the values.  
- EnOceanEnumChannelDescription / EnOceanScaledChannelDescription: define them as subinterfaces of the EnOceanChannelDescription interface. Make the serialization operations generic to the top-level EnOceanChannelDescription interface. Use doubles instead of floats in scaled channels.  
- Remove references to SmartAck and make it clearer that it will not be included for this iteration of the specification.  
- Still keep using only INTERFACES in this specification, but add methods to add/set properties. A bundle that would like to implement a “generic” Device/Message/etc class could then use those methods to do so. |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Revision 8 | July 9th, 2013 | V. Perron, France Telecom Orange  
A. Bottaro, France Telecom Orange  
N. Portinaro, Telecom Italia  
A. Kraft, Deutsche Telekom  
E. Grigorov, Prosyst  
K. Hackbarth, Deutsche Telekom  
  - Rename EnOceanTelegram into EnOceanMessage. Fits better to EnOcean idea of a high-level, multipart message.  
  - Add dBm and redundancy information to EnOceanMessage object. Every EnOceanMessage is sent by burst of three; knowing how many have been actually received, and at which average power level, can help giving an idea of the link quality.  
  - Narrow EnOceanHost’s capabilities to "what should be awaited from a Gateway device" more than "what can ESP do"; we should, as it’s done with Zigbee and the ZCL, not stick to ESP for Gateways, since some hardware vendors would not follow it anyway.  
  - Datafields have been renamed to Channels, to stick better to EnOcean notions. Enumerated channels have been split into Enumerated as before, and a Flag type that describes boolean channels. |
| Revision 9 | July 31st, 2013 | V. Perron, France Telecom Orange  
A. Bottaro, France Telecom Orange  
M. Robin, France Telecom Orange  
  - EXPORT scenario: BD chooses the appropriate dongle, associate service PID and sender ID propose an optional API to retrieve the sender ID or deassociate it.  
  - EnOceanHost: remove ability to send messages (role of the BD) but add an API to retrieve the sender ID associated to a service PID, if allocated within that chip’s ID pool.  
  - Requirements: EnOceanDevice properties such as profile info, security info... MUST be persisted to survive a framework reset; those properties can only be retrieved during an (often manual) teach-in procedure.  
  - EnOceanDevice: for imported devices, there is a CHIP_ID property that is set by the BD. For exported devices, there is no such property, but an ENOECEAN_EXPORT property is there. In both cases, a SERVICE_PID property is present and unique. |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Revision 10  | Aug. 17th, 2013  | V. Perron, France Telecom Orange  
A. Bottaro, France Telecom Orange  
- Every reference to RMCC has been removed, in favour of a united RPC notion.  
- Corrected main class diagram to make Set interfaces appear better, remove faulty <<Set>> notion.  
- Add the notion of EnOceanChannelDescriptionSet, EnOceanRPCSet, EnOceanMessageSet in Entities  
- Clarify the Operations Summary section: full update.  
- Clarify and rewrote EnOceanDevice section; move 'Export' section there and merge its information.  
- Rewrite EnOceanMessage section, cleanup artifacts from previous versions.  
- Reword the EnOceanChannel section; remove the notion of Shortcut and Friendly Name, those are not standard nor used;  
- Corrected the EnOceanChannelDescription part deeply; now more precise about description sets, new class diagram, introduced the Flag channel better, cleaned up outdated examples, introduce a unique identifier that is to be set for every Description class.  
- Rewrote the EnOcean Remote Management part; is clearer and standardized EnOceanRPCHandler into an EnOceanResponseHandler, with a notifyResponse() handler.  
- Confirm that there is no generic deserialization of the EnOceanRPC byte[] payload as of this specification, since EnOcean remote management is still extremely rare and there is no actual specification of it yet.  
- Moved EnOcean Networking and Security sections to a new “Annex” section at the end of the document. |
| Revision 11  | Aug. 24th, 2013  | V. Perron, France Telecom Orange  
A. Bottaro, France Telecom Orange  
- Remove the getValue() / setValue() interfaces from the EnOceanChannel object, will rely on EnOceanChannelDescription only. Keep get/setRawValue().  
- List of paired devices is an ID-based list.  
- List of available RPC is an ID-based list.  
- Introduce the ‘EXTRA’ subtype of an EnOcean message, which in some rare cases is required to further uniquely identify the message type in a Set.  
- The securityLevelFormat property of a device cannot be stored as a service property since it is discovered later on with a dedicated teach-in message. Same goes for non-essential properties such as the Name and ProfileName of the device.  
- Every registered Set object must provide a PROVIDER_ID and VERSION identifiers to not be in conflict with others. No more constraints are specified.  
- An EnOceanHost object is registered as a Host, but not as a device: it bears no profile information.  
- Add details about the format and constraints of EnOceanChannelDescription unique Ids. |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Revision 12 | Aug 14th, 2013 | V. Perron, France Telecom Orange  
A. Bottaro, France Telecom Orange  
- Teach-in standard procedure may include Manufacturer ID, it is now a registered property of EnOceanDevice.  
- Emphasize the "silent dropping" behaviour of the BaseDriver for any message coming from an unknown peer.  
- Filters and device properties confirmed as String objects. No embedded objects in Event Admin. |
| Revision 13 | Oct 10, 2013 | V. Perron, France Telecom Orange  
A. Bottaro, France Telecom Orange  
N. Portinaro, Telecom Italia  
E. Grigorov, Prosyst  
- Introduce EnOceanMessageDescriptionSet interface.  
- EnOceanMessageSet: PROVIDER_ID and VERSION registration properties MAY be specified.  
- The EnOceanChannel.getDescription() has been pulled off, and we'll rely on the service registry to get them.  
- Add more details concerning Event Admin topics and properties, specially, how Generic Messages may be sent over without any message description registered.  
- EnOceanMessage.setStatus() method removed, since the status of a message is an instance-constant.  
- EnOceanMessage.deserializel() actually uses data bytes from the EnOcean Serial packet in order to include signal strength information, repeating status, and so on.  
- EnOceanMessages may need an "extra" identifier from time to time, apart from the ROFG-FUNC-TYPE. Sometimes it will be the "direction" parameter, or something else. |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Revision 14 | Nov 12, 2013 | V. Perron, France Telecom Orange  
A. Bottaro, France Telecom Orange  
N. Portinaro, Telecom Italia  
E. Grigorov, Prosyst  
M. Hönsch, EnOcean  
- Remove the PROVIDER_ID/VERSION identifiers from all Sets.  
- EnOceanDataChannelDescription: input DOMAIN and output RANGES, mathematics standard denominations.  
- EnOceanDevice: remove NAME & PROFILE_NAME service properties.  
- EXPORT situation: documentation update and clarifications.  
- Updates to EnOceanDevice's send method, renamed to invoke and used solely for EnOceanRPC.  
- Introduction of enocean.device.export property on Event Admin to carry exported device's messages onto EnOcean network.  
- Rename EnOceanResponseHandler to EnOceanHandler, keep only the RPC return method.  
- EnOceanDevice supports external modification of profile for manual input, necessary for some devices. |
| Revision 15 | March 13, 2014 | A. Bottaro, France Telecom Orange  
- The javadoc is inserted into the RFC document (Section n°7).  
- The new class diagram following OSGi conventions is now in the RFC document (Fig 3).  
- “Service” diagrams are illustrating the operation summary. Fig 4 depicts the device ‘import’ situation, Fig 5 depicts the device export situation, Fig 6 depicts the use of description set services.  
- A schema showing the technical layers for an application to access EnOcean devices with potential refinement drivers implementing the EnOcean abstraction layer called EnOcean Equipment Profiles (Fig 7) in EnOcean Base Driver and/or a technology agnostic abstraction layer (e.g., implementing RFC 196). |
| Revision 16 | June 5, 2014  | A. Chazalet, Orange  
A. Bottaro, Orange  
- Update the document, fix links, fix typos, etc. |
1 Introduction

EnOcean is a standard wireless communication protocol designed for low-cost and low-power devices by EnOcean Alliance.

EnOcean is widely supported by various types of devices such as smart meters, lights and many kinds of sensors in the residential area. OSGi applications need to communicate with those EnOcean devices. This specification defines how OSGi bundles can be developed to discover and control EnOcean devices on the one hand, and act as EnOcean devices and interoperate with EnOcean clients on the other hand. In particular, a Java mapping is provided for the standard representation of EnOcean devices called EnOcean Equipment Profile.

The specification also describes the external API of an EnOcean Base Driver according to Device Access specification, the example made by ZigBee Device Service specification and spread OSGi practices on residential market.

2 Application Domain

System Architecture

When installing a new EnOcean network into a residential network with an OSGi home gateway, there are 2 options:

- Add EnOcean communication capability to your home gateway, with an additional hardware such as a USB device called "dongle" and then add the necessary software (bundles) to interpret the EnOcean messages.
- Replace the current home gateway with one featuring EnOcean communication.

In both cases OSGi applications call the EnOcean driver API to communicate with the EnOcean devices as shown in Figure 1.
The EnOcean specification defines two main types of devices: the transmitters and the receivers. Some receivers can be used as repeaters, and therefore are using bidirectional communication. The transmitters are using unidirectional communication only.

The very recent ‘Smart Ack’ specification now enables transmitters to stay active for a few milliseconds after a transmission in order to receive messages from a remote device. For this to be possible, “mailboxes” have to be enabled on line-powered devices.

The EnOcean network is mainly composed of those transmitters paired to receivers through a “teach-in” procedure. It is a many-to-many model with no particular hierarchy, the opposite of a star network like Zigbee where every device relies on a single coordinator.

In this respect, the EnOcean gateway’s hardware is no more and no less than a universal EnOcean transceiver, for which the “teach-in” and control procedures have to be software-defined.

**EnOcean Stack**

The EnOcean stack is shown in Figure 2. The three bottom layers, the PHYSICAL layer (not shown in the figure), the DATALINK layer and the NETWORK layer are defined by the ISO/IEC14543-3-10 standard, which is a new standard for the wireless application with ultra-low power consumption.

The EnOcean standard defines the Application and Security layers; it also defines:

1. The EnOcean Serial Protocol (ESP) for serial communication between a host and capable EnOcean modules;
2. The EnOcean Radio Protocol (ERP) defines packeted radio communication between EnOcean nodes;
3. Smart-Ack describes the use of “Mailboxes” on line-powered devices to send messages to energy-harvesting transmitters;
4. The EnOcean Equipment Profiles (EEP), described in detail in the next section, defines standard device profiles to be used by EnOcean devices.

The ISO standard enabled the physical, data link and network layers to be available for all, while the EnOcean application layers are available by joining the EnOcean Alliance.
EnOcean Equipment Profiles (EEP)

The EnOcean Equipment Profiles enables interoperability between products developed by different vendors. For example, in a light control scenario, switches developed by a vendor can turn on and turn off lights developed by another vendor if both vendors are aware of each others EEP Profiles. The EnOcean Alliance draws up the specifications for the applications based on the standard.

A device’s EEP profile is fully defined by three combined elements:

1. Its EnOcean Radio Protocol radio message type (RORG, 8 bits)
   - Basic functionality of the data content (FUNC, 6 bits)
   - Type of device; it refines the main functionality given by FUNC (TYPE, 7 bits)
   - Extra information: in a few rare cases, messages are not entirely defined by those three identifiers, and some extra information is needed, such as a ‘DIRECTION’ or ‘CATEGORY’ information.

There are currently around 100 profiles defined.

When the existing profiles are not adequate, it is possible to create a new profile. Once developed, it should be submitted to the technical working group of the EnOcean Alliance.
3 Problem Description

With the increasing number of EnOcean vendors, the number of manufacturer-specific APIs is also raising, causing the following problems:

- Application developers cannot rely on standard EnOcean hardware interoperability within the target residential gateway’s environment.
- An application that was developed for a given environment may not work in other environments without significant changes.

Those problems make it difficult for third parties to develop portable OSGi applications communicating with EnOcean devices.

The standard Java API requested in this RFC for the access of EnOcean devices would give developers a unified way of communicating with EnOcean devices, allowing developers to rely on a single, vendor-agnostic API.

4 Requirements

R1: The solution MUST provide an API for controlling EnOcean devices.

R2: The solution MUST provide a base driver interface as an OSGi service for the following operations: device and service discovery, network management, binding management, device management.

R3: The solution SHOULD enable applications to trigger a re-scan of the network to refresh the registry with actual EnOcean device services.

R4: The solution MUST provide a mechanism which notifies OSGi applications of events occurred in the EnOcean network and devices.

R5: The solution MUST register a Device Service object representing each found EnOcean device into Service Registry and unregister the Device Service object when the EnOcean device is unavailable or has not sent updates since a very long time.

R6: The solution MUST associate an EEP profile for each found EnOcean device and update the EEP if it is changing.

R7: The solution MUST be able to add new profiles to the existing ones (in the case of a new profile is created by a member of the EnOcean Alliance).
**R8**: The solution *MAY* define the driver provisioning process in accordance with the OSGi Device Access specification.

**R9**: The solution *MUST* be independent from the physical interface used to control the EnOcean network. The solution *MUST* likewise work with network controllers based on EnOcean built-in chips, EnOcean USB dongles and high level protocols offered by EnOcean Gateway Devices compliant with the EnOcean Alliance specification.

**R10**: The solution *MUST* include device access control based on user and application permissions compliant with the OSGi security model.
5 Technical Solution
6 Initial Spec Chapter

Introduction
EnOcean is a standard wireless communication protocol designed for low-cost and low-power devices by EnOcean Alliance. The protocol is widely supported by various types of devices such as lights and many kinds of sensors in the residential area. OSGi residential applications need to communicate with those EnOcean devices. This specification defines how OSGi bundles can be developed to discover and control EnOcean devices on the one hand, and act as EnOcean devices and interoperate with EnOcean clients on the other hand. In particular, a Java mapping is provided for the standard representation structure of EnOcean devices, i.e. the organizational description of all EnOcean Equipment Profiles (EEPs) [5].

Specified APIs are meant to be implemented by an EnOcean Base Driver. According to Device Access specification terminology, a base driver mirrors the network activity by registering and unregistering services representing available EnOcean devices. Those services are named EnOcean device services.

Essentials
- **Scope** – This specification is limited to general device discovery and control aspects of the standard EnOcean specifications. Aspects concerning the representation of specific or proprietary EnOcean profiles is not addressed.
- **Transparency** – EnOcean devices discovered on the network and devices locally implemented on the platform are represented in the OSGi service registry with the same API.
- **Lightweight implementation option** – The full description of EnOcean device services on the OSGi platform is optional. Some base driver implementations may implement all the classes including EnOcean device description classes while implementations targeting constrained devices are able to implement only the part that is necessary for EnOcean device discovery and control.
- **Network Selection** – It must be possible to restrict the use of the EnOcean protocols to a selection of the connected devices.
- **Event handling** – Bundles are able to listen to EnOcean events.
- **Discover and control EnOcean devices as OSGi services** – Available learnt (via an EnOcean teach-in procedure) EnOcean external endpoints are dynamically reified as OSGi services on the service registry upon discovery.
- **OSGi services as exported EnOcean devices** – OSGi services implementing the API defined here and explicitly set to be exported should be made available to networks with EnOcean enabled endpoints in a transparent way.
Entities

- **EnOcean Base Driver** – The bundle that implements the bridge between OSGi and EnOcean networks (see Figure 3). It is responsible for accessing the various EnOcean gateway chips on the execution machine, and ensures the reception and translation of EnOcean messages into proper objects. It is also used to send messages on the EnOcean network, using whatever chip it deems most appropriate.

- **EnOcean Host** – The EnOceanHost object is a link between the software and the EnOcean network. It represents the chip configuration (gateway capabilities) described in ESP3[9]. It is registered as an OSGi service.

- **EnOcean Device** – An EnOcean device. This entity is represented by a EnOceanDevice interface and registered as a service within the framework. It carries the unique chip ID of the device, and may represent either an imported or exported device, which may be a pure transmitter or a transceiver.

- **EnOcean Message** – Every EnOcean reporting equipment is supposed to follow a “profile”, which is essentially the way the emitted data is encoded. In order to reflect this standard as it is defined in the EEP[5], manufacturers are able to register the description of “Messages”, the essence of a profile, along with their associated payload (as Channels). See “EnOcean Channels” below for more information.

- **EnOcean Channel** – EnOcean channels are available as an array inside EnOceanMessage objects. They are a useful way to define any kind of payload that would be put inside of an EnOcean Message.

EnOcean Messages and their associated Channels can be described with EnOceanMessageDescription and EnOceanChannelDescription interfaces. Description providers aggregate these descriptions in sets that they register with EnOceanMessageDescriptionSet and EnOceanChannelDescriptionSet interfaces within the framework.

The mechanism allows in particular a lightweight implementation of the EnOcean device service platform, by leaving the possibility not to implement the unnecessary message or channel descriptions.

- **EnOcean RPC** – An interface that enables the invocation of vendor-specific Remote Procedure Calls and Remote Management Commands. These are particular types of Messages and are not linked to any EnOcean Profile, so that their descriptions are defined and registered in another way. The RPCs are documented via the EnOceanRPCDescription objects gathered into registered EnOceanRPCDescriptionSet services.

- **EnOcean Handler** – Enables clients to asynchronously get answers to their RPCs.

- **EnOcean Client** – An application that is intended to control EnOcean device services.

- **EnOcean Exception** – Delivers errors during EnOceanMessage serialization/deserialization or during execution outside transmission.
Figure 3: EnOcean Service Specification class diagram
Operation Summary
To make an EnOcean device service available to EnOcean clients on the OSGi platform, it must be registered under the EnOceanDevice interface within the OSGi framework.

The EnOcean Base Driver is responsible for mapping external devices into EnOceanDevice objects, through the use of an EnOcean gateway. The latter is represented on OSGi framework as an object implementing EnOceanHost interface. EnOcean “teach-in” messages will trigger this behavior, this is called a device import situation (see Figure 4).

Client bundles may also expose framework-internal (local) EnOceanDevice instances, registered within the framework (see Figure 5). The Base Driver then should emulate those objects as EnOcean devices on the EnOcean network. This is a device export situation, made possible by the use of the 127 virtual base IDs available on an EnOcean gateway. For more information about this process, please report to the “Exporting an EnOcean device” section below.

EnOcean clients send RPCs (Remote Procedure Calls) to EnOcean devices and receives RPC responses and messages from them. Messages coming from EnOcean devices are accessible through Event Admin.

RPCs and messages content are specified by EnOcean Alliance or vendor-specific descriptions. Those descriptions may be provided on the OSGi platform by any bundle through the registration of EnOceanRPCDescriptionSet.
EnOceanMessageDescriptionSet and EnOceanChannelDescriptionSet services. Every service is a set of description that enables applications to retrieve information about supported RPCs, messages or channels that compose messages.

![Diagram showing EnOceanMessageDescriptionSet and EnOceanChannelDescriptionSet services]

**EnOcean Base Driver**

Most of the functionality described in the operation summary is implemented in an EnOcean base driver. This bundle implements the EnOcean protocol and handles the interaction with bundles that use the EnOcean devices. An EnOcean base driver is able to discover EnOcean devices on the network and map each discovered device into an OSGi registered EnOceanDevice service. It is also the receptor, through EventAdmin service and OSGi service registry, of all the events related to local devices and clients. It enables bidirectional communication for RPC and Channel updates.

Several base drivers may be deployed on a residential OSGi device, one for every supported network technology. An OSGi device abstraction layer may then be implemented as a layer of refinement drivers above a layer of base drivers. The refinement driver is responsible for adapting technology-specific device services registered by the base driver into device services of another model (see AbstractDevice interface in Figure 7). In the case of a generic device abstraction layer, the model is agnostic to technologies.

The EnOcean Alliance defines their own abstract model with EnOcean Equipment Profiles and refinement drivers may provide the implementation of all EEPs with EnOcean specific Java interfaces. The AbstractDevice interface of Figure 7 is then replaced by an EEP specific Java interface in that case. The need and the choice of the abstraction depends on the targeted application domain.
**EnOcean Host**

The EnOcean host represents an EnOcean gateway chip. Any EnOcean device service implementation should rely on at least one Gateway Chip in order to send and receive messages on the external EnOcean network. This interface enables standard control over an EnOcean compatible chip. Every `EnOceanHost` object should at least be identified by its unique chip ID.

The `EnOceanHost` interface enables OSGi applications to:

- Get or set gateway metadata (version, name, etc);
- Reset the gateway chip device;
- Retrieve a chip ID (derived from EnOcean’s BASE_ID) for the given Service PID of a device.

**EnOcean Device Generics**

A physical EnOcean device is reified as an `EnOceanDevice` object within the framework. Any `EnOceanHost` is also an `EnOceanDevice`, but the two concepts are not linked by any inheritance.

An EnOcean device holds most of the natural properties for an EnOcean object: its unique ID, the profile, a friendly name, its security information, and its available RPCs – along with the associated getters (and setters when applicable). All those properties MUST be persistent across restart so that teach-in procedures are made only once.

It also holds methods that reflect the natural actions a user application may physically trigger on such a device: send a message to the device, send a teach-in message to the device, or switch the device to learning mode.

Every EnOcean Device keeps a service PID property that is assigned either by the base driver or by any service-exporting bundle. The property value format is free and the value must be unique on the framework.

The properties on which EnOceanDevice services can be filtered on are: the device’s service PID and chip ID, and its profile identifiers (RORG / FUNC / TYPE integers).

The `EnOceanDevice` also keeps security features as defined in the EnOcean Security Draft[9], which allow for a security level format (integer mask), an encryption key and/or a rolling authentication code. See the Security Of EnOcean networks section below.

The `EnOceanDevice` service MUST also be registered with `org.osgi.service.device.Constants.DEVICE_CATEGORY` property (see OSGi Compendium: 103 Device Access Specification) that describes a table (String[]) of categories to which the device belongs. One value MUST be “EnOcean” (`org.osgi.service.enocean.EnOceanDevice.DEVICE_CATEGORY`).

The additional properties (defined in Device Access – 103.2.1): DEVICE_DESCRIPTION, DEVICE_SERIAL values are not specified here as no description nor application-level serial number are provided in the EnOcean standard protocol.

**Import Situation**

In *import* situations, the device’s chip ID is uniquely set by the Base Driver, according to the one present in the teach-in message that originated the Device’s creation. The service PID (cf. Core Specification R4 v4.3, section 5.2.5) should also be generated and deterministically derived from the chip ID to allow reconstruction of a device without a new teach-in process after a framework restart.
Export Situation

In export situations:

1. The registering Client bundle sets the service PID of the EnOceanDevice object by itself, in a unique manner, and registers that object.

2. The chip ID (this device’s EnOcean source ID when it issues messages) will be allocated by the Base Driver. The latter keeps a dictionary of the currently allocated chip IDs. The Client bundle must also set an ENOCEAN_EXPORT property in the registered device’s Property Map.

The standard way to programmatically retrieve an exported chip ID from a given service PID is by using EnOceanHost’s dedicated interface for this use.

The Base Driver MUST ensure the persistence of the CHIP_ID:SERVICE_PID mapping.

As an application developer, please refer to the documentation of your Base Driver to know its policies concerning exported chip ID updating, deletion and exhaustion.

Interface

The EnOceanDevice interface enables client bundles to:

- Get or set the security features of the device in a protected way;
- Retrieve the currently paired devices in the case of a receiver, as a collection of device IDs;
- Get the ID-based list of currently available RPCs for the device, as a Map of {manufacturerID: [functionId1, functionId2, ...]};
- Invoke RPCs onto the device, through the invoke() call.

EnOcean Messages

Introduction

EnOcean Messages are at the core of the EnOcean application layer as a whole and the EnOcean Equipment Profile specification[5], in particular. Every exchange of information within EnOcean networks is done with a dedicated message. In this specification we will be especially interested in a particular portion of an EnOcean Serial Protocol Type 1 (RADIO) message:
This model enables reading both the EnOcean radio telegram data and the associated metadata that may be attached to it in a single object, EnOceanMessage.

In case the 'Optional Data' section gets missing at the lowest level (the radio access layer not following ESP protocol for instance) it is the responsibility of the Base Driver to mock the missing field's (dBm, destinationID, …) values.

**Mode of operation**

Any EnOceanMessage object creation will be mirrored to Event Admin.

Details about the available topics, filters and properties can be found in the Event API section below.

EnOceanMessage objects will be created only if the originating device has already been registered in the OSGi Service Registry, along with profile information.

**Identification**

The RORG of a message defines its shape and generic type; all the RORGs are defined in the EnOcean Radio Specification.

An addressed message will be encapsulated into an Adressed Telegram (ADT) by the base driver transparently; this means that from the application level, it will be represented under its original RORG, but with a valid destinationID.

A particular EnOcean Equipment Profile message is identified by three numbers: its RORG, and its FUNC, TYPE and EXTRA subtypes. In EnOcean, a (RORG, FUNC, TYPE) triplet is enough to identify a profile; though an EXTRA identifier is sometimes needed to identify a particular message layout for that profile.

Those identifiers allow for retrieving EnOceanMessageDescription objects within a registered EnOceanMessageDescriptionSet, which give the application more information to parse the message.

**Interface**

The methods available in the EnOceanMessage interface are:

- Identification methods, retrieving the message’s profile, sender ID, optional destination ID, status;
- A method to get the raw bytes of payload data in the message. This data can then be passed to the
Deserializer of the EnOceanMessageDescription object to be converted to EnOceanChannels, which may -again- be documented (through EnOceanChannelDescription objects) or not.

- Link quality information read-only methods that mirror some of the 'Optional Data' header information.

### EnOcean Message Description

**EnOceanMessageDescription** objects exposes only two methods:

- **deserialize()**: makes the user able to deserialize the payload bytes of a raw EnOceanMessage object, into a collection of EnOceanChannel objects.
- **serialize()**: serializes the input EnOceanChannel objects into a collection of bytes.

### EnOcean Channel

The **EnOceanChannel** interface is the first step of an abstraction to generate or interpret EnOceanMessage channels with plain Java types.

The simple **EnOceanChannel** interface provides a way to separate the different fields in a message payload, knowing their offset and size in the byte array that constitutes the full message's payload.

At the **EnOceanChannel** level, the only way to get/set the information contained in the channel is through a pair of **getRawValue()** and **setRawValue()** methods, which act on plain bytes.

Those bytes are meant right-aligned, and the number of those bytes is the size of the datafield, floored up to the next multiple of 8. For instance, a 3-bit long channel would be encoded on one byte, all the necessary information starting from bit 0.

Every EnOceanMessage as described in the EEP Specification contains a various amount of channels, each of them being identified by their unique ID.

This ID, or **channelID**, is constituted of the “Shortcut” field of this channel from the EEP 2.5 Specification[5], and a number fixed by the order of appearance of such a “Shortcut” in the specification.

This unique identifier links a Channel to an **EnOceanChannelDescription** object that provides more information to encode and decode that channel’s information; see below for more details. This enables for loose coupling of the raw Channel itself and a richer, 3rd-party provided, information.

As an example, if the platform being developed is an electronic display that waits for Messages from a well-known temperature sensor, the Client bundle on the platform may interpret the Temperature Channels in every Temperature Message without needing an appropriate **TemperatureChannelDescription** object; it may directly cast and convert the Byte[] array of every received message to a properly valued Double and display that.

Otherwise, it could as well use the **channelID** to get a **TemperatureChannelDescription** object that would properly handle the deserialization process from the raw bytes to a proper, physical unit-augmented, result.
EnOcean Channel Description

The EnOceanChannelDescription interface enables the description of all the various channels as specified in the EnOcean specification, as well as the description of channels issued by 3rd party actors.

Those description objects are retrieved from the registered EnOceanChannelDescriptionSet interface using a unique ID known as the channelID.

Here are the Channel types defined in this specification:

- **CHANNEL_TYPE_RAW**: A collection of bytes. This type is used when the description is not provided, and is thus the default. For this type, the EnOceanChannelDescription's deserialize() call actually returns a byte[] collection. The encryption key or a device ID on 4 bytes are examples of such raw types.

- **CHANNEL_TYPE_DATA**: A scaled physical value. Used when the data can be mapped to a physical value; for instance, the 'WND – Wind Speed' channel is a raw binary value, in a range from 0 to 255, that will be mapped as a wind speed between 0 and 70 m/s. For this type, the EnOceanChannelDescription's deserialize() call actually returns a Double value.

- **CHANNEL_TYPE_FLAG**: A boolean value. Used when the Channel value can be either 1 or 0. The “Teach-In” Channel is a well-known example; this 1-bit field may either be 0 or 1, depending whether the Message is a teach-in one or not. For this type, the EnOceanChannelDescription's deserialize() call actually returns a Boolean value.

- **CHANNEL_TYPE_ENUM**: An enumeration of possible values. Used when the Channel can only take a discrete number of values. More complicated than the Flag type, Enumerated types may have thresholds: for instance, the A5-30 “Digital Input- Input State (IPS)” channel is an 8-bit value which means “Contact closed” between 0 and 195, and “Contact open” from 196 to 255. For this type, the EnOceanChannelDescription's deserialize() call actually returns an EnOceanChannelEnum object.

According to the channel type, the actual description object should implement one of the following specialized interfaces. This will ease the use of casting to the specialized interfaces on documented channels.

EnOcean Data Channel Description

The EnOceanDataChannelDescription interface inherits from EnOceanChannelDescription interface.

Two more methods give access to the integer input domain of the data channel (such as 0-255) and to the floating-point output range of it (such as -30.0°C – 24.5°C). A method is also present to retrieve the physical unit of the
channel. The serialize() and deserialize() methods are implemented to easily convert from the raw byte[] collection to a Double, and vice-versa.

Here are a few samples of such Channels:

<table>
<thead>
<tr>
<th>Short</th>
<th>Description</th>
<th>Possible implemented name</th>
<th>Domain</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP</td>
<td>Temperature (linear)</td>
<td>TemperatureScaledChannel_X</td>
<td>0..255</td>
<td>-10°..+30°</td>
<td>°C</td>
</tr>
<tr>
<td>HUM</td>
<td>Humidity (linear)</td>
<td>HumidityScaledChannel_X</td>
<td>0..250</td>
<td>0..100</td>
<td>%</td>
</tr>
</tbody>
</table>

**EnOcean Flag Channel Description**

The EnOceanFlagChannelDescription interface inherits from the EnOceanChannelDescription interface.

Those channels, are typically used for On/Off reporting values (like a switch); they have no additional methods, though the deserialize() method converts the input bit into a proper Boolean object.

**EnOcean Enumerated Channel Description**

The EnOceanEnumChannelDescription interface inherits from the EnOceanChannelDescription interface.

The additional method provided to this interface is getPossibleValues(), which returns an array of the available EnOceanChannelEnumValue objects accessible to this channel. Every EnumValue object contains its integer input range and a String identifier that defines its meaning.

The serialize() and deserialize() methods of an EnOceanEnumChannelDescription object thus convert an integer input value (say, 156) to an EnOceanChannelEnumValue, and vice-versa.

Here is an example that shows the input range and the associated EnOceanChannelEnumValues:

<table>
<thead>
<tr>
<th>Device profile</th>
<th>EnOceanChannelEnumValue</th>
<th>Start</th>
<th>Stop</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fan speed stage switch</td>
<td>FanStageSwitch_Stage3</td>
<td>0</td>
<td>144</td>
<td>Fan speed: Stage 3</td>
</tr>
<tr>
<td>Fan speed stage switch</td>
<td>FanStageSwitch_Stage2</td>
<td>145</td>
<td>164</td>
<td>Fan speed: Stage 2</td>
</tr>
<tr>
<td>Fan speed stage switch</td>
<td>FanStageSwitch_Stage1</td>
<td>165</td>
<td>189</td>
<td>Fan speed: Stage 1</td>
</tr>
<tr>
<td>Fan speed stage switch</td>
<td>FanStageSwitch_Stage0</td>
<td>190</td>
<td>209</td>
<td>Fan speed: Stage 0</td>
</tr>
</tbody>
</table>

**EnOcean Remote Management**

Remote Management is a feature which allows EnOcean devices to be configured and maintained over the air using radio messages.

The Remote Procedure Calls, or RPCs - as defined by the EnOcean Remote Management specification[6].- are not related to any EnOcean Equipment Profile.

**EnOcean RPC**

An EnOceanRPC object enables client bundles to remotely manage EnOcean devices using already defined behavior.

RPCs are mandatorily defined by a MANUFACTURER_ID (11 bits, 0x7FF for the EnOcean alliance) and a unique FUNCTION_ID code on 12 bits.

RPCs are called directly onto an EnOceanDevice object via the device.invoke() method, which accepts also a non-mandatory EnOceanHandler object as a parameter to retrieve the asynchronous answer.

Broadcasted RPCs can be addressed directly to the Base Driver using the relevant Event Admin topic; see the Event API section below.
EnOcean Handler

Responses to RPCs are processed by the driver and sent back to a handler using `EnOceanHandler.notifyResponse()` method when an `EnOceanHandler` is passed to the base driver.

Working With an EnOcean Device

Service Tracking

All discovered EnOcean devices in the local networks are registered under `EnOceanDevice` interface within the OSGi framework. Every time an EnOcean device appears or quits the network, the associated OSGi service is registered or unregistered in the OSGi service registry. Thanks to the EnOcean Base Driver, the OSGi service availability in the registry mirrors EnOcean device availability on EnOcean network.

Thanks to service events, a bundle is able to track the addition, modification and removal of an `EnOceanDevice` service.

Below stands an example showing how the tracking can be implemented. The sample Controller class extends the `ServiceTracker` class so that it can track all `EnOceanDevice` services.

```java
public class Controller extends ServiceTracker {
    public Object addingService(ServiceReference arg0) {
        Object service = context.getService(arg0);
        if (service != null && service instanceof EnOceanDevice) {
            eoDevice = (EnOceanDevice) service;
        }
        Logger.debug(service.getClass().getName() + " service found.", null);
        return service;
    }

    public void modifiedService(ServiceReference arg0, Object arg1) {
        /* Unimplemented */
    }

    public void removedService(ServiceReference arg0, Object service) {
        if (service instanceof EnOceanDevice) {
            eoDevice = null;
            Logger.debug("EnOceanDevice service was removed.", null);
        }
    }
}
```

Event API

EnOcean events must be delivered to the `EventAdmin` service by the EnOcean implementation, if present. EnOcean event topic follow the following form:

```
org.osgi.service.enocean/EnOceanEvent/SUBTOPIC
```

Here are the available subtopics:

**MESSAGE_RECEIVED**

**Properties**: (every event may dispatch some or all of the following properties)

- `EnOceanDevice.CHIP_ID` – (enocean.device.chip_id/String). The chip ID of the sending device.
- `ConstantsSERVICE_PID` – (service.pid/String). The service PID of the exported device.
- `EnOceanDevice.RORG` – (enocean.device.profile.rorg/String). The RORG (Radio Telegram Type) of the sending device.
EnOcean Device Service Specification

Final

June 5, 2014

- **EnOceanDevice.FUNC** – (enoecean.device.profile.func/\String). The FUNC profile identifier of the sending device.

- **EnOceanDevice.TYPE** – (enoecean.device.profile.type/\String). The TYPE profile identifier of the sending device.

- **EnOceanEvent.PROPERTY_MESSAGE** – (enoecean.message/EnOceanMessage). The EnOceanMessage object associated with this event.

- **EnOceanEvent.PROPERTY_EXPORTED** – (enoecean.message.is_exported/Object). The presence of this property means that this message has actually been exported from a locally implemented EnOcean Device.

**RPC_BROADCAST**

This event is used whenever an RPC is broadcasted on EnOcean networks, in IMPORT or EXPORT situations.

Properties: (every event may dispatch some or all of the following properties)

- **EnOceanRPC.MANUFACTURER_ID** – (enoecean.rpc.manufacturer_id/\String). The RPC's manufacturer ID.

- **EnOceanRPC.FUNCTION_ID** – (enoecean.rpc.function_id/\String). The RPC's function ID.

- **EnOceanEvent.PROPERTY_EXPORTED** – (enoecean.message.is_exported/Object). The presence of this property means that this RPC has actually been exported from a locally implemented EnOcean Device.

- **EnOceanEvent.PROPERTY_RPC** – (enoecean.rpc/EnOceanRPC). The EnOceanRPC object associated with this event.

**EnOcean Exceptions**

The EnOceanException can be thrown and holds information about the different EnOcean layers. Here below, ESP stands for “EnOcean Serial Protocol”. The following errors are defined:

- **FAILURE** – (0x01) Operation was not successful.

- **ESP_RET_NOT_SUPPORTED** – (0x02) The ESP command was not supported by the driver.

- **ESP_RET_WRONG_PARAM** – (0x03) The ESP command was supplied wrong parameters.

- **ESP_RET_OPERATION_DENIED** – (0x04) The ESP command was denied authorization.

- **INVALID_TELEGRAM** – (0xF0) The message was invalid.

---

**7 Javadoc**
## Package org.osgi.service.enocean

### Interface Summary

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnOceanChannel</td>
<td>32</td>
</tr>
<tr>
<td>EnOceanDevice</td>
<td>34</td>
</tr>
<tr>
<td>EnOceanHandler</td>
<td>44</td>
</tr>
<tr>
<td>EnOceanHost</td>
<td>45</td>
</tr>
<tr>
<td>EnOceanMessage</td>
<td>49</td>
</tr>
<tr>
<td>EnOceanRPC</td>
<td>52</td>
</tr>
<tr>
<td>EnOceanSerialInOut</td>
<td>54</td>
</tr>
</tbody>
</table>

**EnOceanChannel**
Holds the raw value and channel identification info of an EnOceanChannel.

**EnOceanDevice**
This interface represents a physical device that communicates over the EnOcean protocol.

**EnOceanHandler**
The interface used to get callback answers from a RPC or a Message.

**EnOceanHost**
This interface represents an EnOcean Host, a device that offers EnOcean networking features.

**EnOceanMessage**
Holds the necessary methods to interact with an EnOcean message.

**EnOceanRPC**
A very basic interface for RPCs.

**EnOceanSerialInOut**
This class is intended to provide in/out methods from serial to emulate an EnOcean dongle during integration tests.

### Class Summary

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnOceanEvent</td>
<td>40</td>
</tr>
</tbody>
</table>

### Exception Summary

<table>
<thead>
<tr>
<th>Exception Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnOceanException</td>
<td>41</td>
</tr>
</tbody>
</table>

This class contains code and definitions necessary to support common EnOcean exceptions.
public interface EnOceanChannel

Holds the raw value and channel identification info of an EnOceanChannel.

Version: 1.0

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getChannelId()</td>
<td>Returns: The unique ID of this channel.</td>
</tr>
<tr>
<td>int getOffset()</td>
<td>Returns: The offset, in bits, where this channel is found in the telegram.</td>
</tr>
<tr>
<td>byte[] getRawValue()</td>
<td>Gets the raw value of this channel.</td>
</tr>
<tr>
<td>int getSize()</td>
<td>Returns: The size, in bits, of this channel.</td>
</tr>
<tr>
<td>void setRawValue(byte[] rawValue)</td>
<td>Sets the raw value of a channel.</td>
</tr>
</tbody>
</table>

Method Detail

getChannelId

String getChannelId()

getOffset

int getOffset()

getSize

int getSize()

getRawValue

byte[] getRawValue()

Gets the raw value of this channel.
setRawValue

void setRawValue(byte[] rawValue)

Sets the raw value of a channel.
public interface EnOceanDevice

This interface represents a physical device that communicates over the EnOcean protocol.

Version:
1.0

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIP_ID</td>
<td>Property name for the mandatory CHIP_ID of the device</td>
<td>35</td>
</tr>
<tr>
<td>DEVICE_CATEGORY</td>
<td>Property name for the mandatory DEVICE_CATEGORY of the device</td>
<td>35</td>
</tr>
<tr>
<td>ENOCEAN_EXPORT</td>
<td>Property name that defines if the device is exported or not.</td>
<td>36</td>
</tr>
<tr>
<td>FUNC</td>
<td>Property name for the radiotelegram functional type of the profile associated with this device.</td>
<td>35</td>
</tr>
<tr>
<td>MANUFACTURER</td>
<td>Property name for the manufacturer ID that may be specified by some teach-in messages.</td>
<td>36</td>
</tr>
<tr>
<td>RORG</td>
<td>Property name for the radiotelegram main type of the profile associated with this device.</td>
<td>35</td>
</tr>
<tr>
<td>SECURITY_LEVEL_FORMAT</td>
<td>Property name for the security level mask for this device.</td>
<td>36</td>
</tr>
<tr>
<td>TYPE</td>
<td>Property name for the radiotelegram subtype of the profile associated with this device.</td>
<td>35</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getChipId()</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>byte[] getEncryptionKey()</td>
<td>Returns the current encryption key used by this device.</td>
<td>38</td>
</tr>
<tr>
<td>int getFunc()</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>int[] getLearnedDevices()</td>
<td>Gets the list of devices the device already has learned.</td>
<td>38</td>
</tr>
<tr>
<td>int getManufacturer()</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>int getRollingCode()</td>
<td>Get the current rolling code of the device.</td>
<td>37</td>
</tr>
<tr>
<td>int getRorg()</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>Map getRPCs()</td>
<td>Retrieves the currently available RPCs to this device; those are stored using their manufacturerId:commandId identifiers.</td>
<td>38</td>
</tr>
<tr>
<td>int getSecurityLevelFormat()</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>int getType()</td>
<td></td>
<td>36</td>
</tr>
</tbody>
</table>
### Interface EnOceanDevice

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void invoke(EnOceanRPC rpc, EnOceanHandler handler)</code></td>
<td>Sends an RPC to the remote device.</td>
</tr>
<tr>
<td><code>void remove()</code></td>
<td>Removes the device from both EnOcean Network and OSGi service platform.</td>
</tr>
<tr>
<td><code>void setEncryptionKey(byte[] key)</code></td>
<td>Sets the encryption key of the device.</td>
</tr>
<tr>
<td><code>void setFunc(int func)</code></td>
<td>Manually sets the EEP FUNC of the device.</td>
</tr>
<tr>
<td><code>void setLearningMode(boolean learnMode)</code></td>
<td>Switches the device into learning mode.</td>
</tr>
<tr>
<td><code>void setRollingCode(int rollingCode)</code></td>
<td>Sets the rolling code of this device.</td>
</tr>
<tr>
<td><code>void setType(int type)</code></td>
<td>Manually sets the EEP TYPE of the device.</td>
</tr>
</tbody>
</table>

### Field Detail

#### DEVICE_CATEGORY

```java
class EnOceanDevice {
    public static final String DEVICE_CATEGORY = "EnOcean";
}
```

Property name for the mandatory Device_CATEGORY of the device

#### CHIP_ID

```java
class EnOceanDevice {
    public static final String CHIP_ID = "enocean.device.chip_id";
}
```

Property name for the mandatory CHIP_ID of the device

#### RORG

```java
class EnOceanDevice {
    public static final String RORG = "enocean.device.profile.rorg";
}
```

Property name for the radiotelegram main type of the profile associated with this device.

#### FUNC

```java
class EnOceanDevice {
    public static final String FUNC = "enocean.device.profile.func";
}
```

Property name for the radiotelegram functional type of the profile associated with this device.

#### TYPE

```java
class EnOceanDevice {
    public static final String TYPE = "enocean.device.profile.type";
}
```

Property name for the radiotelegram subtype of the profile associated with this device.
MANUFACTURER

public static final String MANUFACTURER = "enocean.device.manufacturer"

Property name for the manufacturer ID that may be specified by some teach-in messages.

SECURITY_LEVEL_FORMAT

public static final String SECURITY_LEVEL_FORMAT = "enocean.device.security_level_format"

Property name for the security level mask for this device. The format of that mask is specified in EnOcean Security Draft.

ENOCEAN_EXPORT

public static final String ENOCEAN_EXPORT = "enocean.device.export"

Property name that defines if the device is exported or not. If present, the device is exported.

Method Detail

getChipId

int getChipId()

Returns:
The EnOcean device chip ID.

getRorg

int getRorg()

Returns:
The EnOcean profile RORG.

getFunc

int getFunc()

Returns:
The EnOcean profile FUNC, or -1 if unknown.

getType

int getType()

Returns:
The EnOcean profile TYPE, or -1 if unknown.
Interface EnOceanDevice

getManufacturer

int getManufacturer()

Returns:
The EnOcean manufacturer code, -1 if unknown.

getSecurityLevelFormat

int getSecurityLevelFormat()

Returns:
The EnOcean security level format, or 0 as default (no security)

setFunc

void setFunc(int func)

Manually sets the EEP FUNC of the device.

Parameters:
func - the EEP func of the device;

setType

void setType(int type)

Manually sets the EEP TYPE of the device.

Parameters:
type - the EEP type of the device;

setLearningMode

void setLearningMode(boolean learnMode)

Switches the device into learning mode.

Parameters:
learnMode - the desired state: true for learning mode, false to disable it.

getRollingCode

int getRollingCode()

Get the current rolling code of the device.

Returns:
The current rolling code in use with this device’s communications.
Interface EnOceanDevice

setRollingCode

void setRollingCode(int rollingCode)

    Sets the rolling code of this device.

    Parameters:
        rollingCode - the rolling code to be set or initiated.

getEncryptionKey

byte[] getEncryptionKey()

    Returns the current encryption key used by this device.

    Returns:
        The current encryption key, or null.

setEncryptionKey

void setEncryptionKey(byte[] key)

    Sets the encryption key of the device.

    Parameters:
        key - the encryption key to be set.

getLearnedDevices

int[] getLearnedDevices()

    Gets the list of devices the device already has learned.

    Returns:
        The list of currently learned device's CHIP_IDS.

getRPCs

Map getRPCs()

    Retrieves the currently available RPCs to this device; those are stored using their
    manufacturerId:commandId identifiers.

    Returns:
        A list of the available RPCs, in a Map form.

invoke

void invoke(EnOceanRPC rpc, EnOceanHandler handler)
    throws IllegalArgumentException
**Interface EnOceanDevice**

Sends an RPC to the remote device.

Throws:

*IllegalArgumentException*

---

**remove**

```java
void remove()
```

Removes the device from both EnOcean Network and OSGi service platform.
Class EnOceanEvent

```java
final public class EnOceanEvent
extends Object
```

### Field Summary

<table>
<thead>
<tr>
<th>Static String</th>
<th>PROPERTY_EXPORTED</th>
<th>Property key used to tell apart messages that are exported or imported.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROPERTY_MESSAGE</td>
<td>Property key for the EnOceanMessage object embedded in an event.</td>
</tr>
<tr>
<td></td>
<td>TOPIC_MSG_RECEIVED</td>
<td>Main topic for all OSGi dispatched EnOcean messages, imported or exported.</td>
</tr>
</tbody>
</table>

### Constructor Summary

```java
EnOceanEvent()
```

### Field Detail

**TOPIC_MSG_RECEIVED**

```java
public static final String TOPIC_MSG_RECEIVED = "org/osgi/service/enocean/EnOceanEvent/MESSAGE_RECEIVED"
```

Main topic for all OSGi dispatched EnOcean messages, imported or exported.

**PROPERTY_MESSAGE**

```java
public static final String PROPERTY_MESSAGE = "enoecean.message"
```

Property key for the EnOceanMessage object embedded in an event.

**PROPERTY_EXPORTED**

```java
public static final String PROPERTY_EXPORTED = "enoecean.message.is_exported"
```

Property key used to tell apart messages that are exported or imported.

### Constructor Detail

**EnOceanEvent**

```java
public EnOceanEvent()
```
public class EnOceanException
extends RuntimeException

This class contains code and definitions necessary to support common EnOcean exceptions. This class is mostly used with low-level, gateway-interacting code : EnOceanHost.

Version:
1.0

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static short ESP_RET_NOT_SUPPORTED</td>
<td>42</td>
</tr>
<tr>
<td>Operation is not supported by the target device.</td>
<td></td>
</tr>
<tr>
<td>static short ESP_RET_OPERATION_DENIED</td>
<td>42</td>
</tr>
<tr>
<td>The operation was denied.</td>
<td></td>
</tr>
<tr>
<td>static short ESP_RET_WRONG_PARAM</td>
<td>42</td>
</tr>
<tr>
<td>One of the parameters was badly specified or missing.</td>
<td></td>
</tr>
<tr>
<td>static short ESP_UNEXPECTED_FAILURE</td>
<td>42</td>
</tr>
<tr>
<td>Unexpected failure.s</td>
<td></td>
</tr>
<tr>
<td>static short SUCCESS</td>
<td>42</td>
</tr>
<tr>
<td>SUCCESS status code.</td>
<td></td>
</tr>
</tbody>
</table>

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnOceanException(int errorCode)</td>
<td>43</td>
</tr>
<tr>
<td>Constructor for EnOceanException</td>
<td></td>
</tr>
<tr>
<td>EnOceanException(int errorCode, String errorDesc)</td>
<td>42</td>
</tr>
<tr>
<td>Constructor for EnOceanException</td>
<td></td>
</tr>
<tr>
<td>EnOceanException(String errordesc)</td>
<td>42</td>
</tr>
<tr>
<td>Constructor for EnOceanException</td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>int errorCode()</td>
<td>43</td>
</tr>
<tr>
<td>Constructor for EnOceanException</td>
<td></td>
</tr>
</tbody>
</table>
### Field Detail

**SUCCESS**

```java
class EnOceanException {
    public static final short SUCCESS = 0
}
```

SUCCESS status code.

---

**ESP_UNEXPECTED_FAILURE**

```java
class EnOceanException {
    public static final short ESP_UNEXPECTED_FAILURE = 1
}
```

Unexpected failure(s).

---

**ESP_RET_NOT_SUPPORTED**

```java
class EnOceanException {
    public static final short ESP_RET_NOT_SUPPORTED = 2
}
```

Operation is not supported by the target device.

---

**ESP_RET_WRONG_PARAM**

```java
class EnOceanException {
    public static final short ESP_RET_WRONG_PARAM = 3
}
```

One of the parameters was badly specified or missing.

---

**ESP_RET_OPERATION_DENIED**

```java
class EnOceanException {
    public static final short ESP_RET_OPERATION_DENIED = 4
}
```

The operation was denied.

### Constructor Detail

**EnOceanException**

```java
class EnOceanException {
    public EnOceanException(String errordesc)
}
```

Constructor for EnOceanException

Parameters:
- `errordesc` - exception error description

---

**EnOceanException**

```java
class EnOceanException {
    public EnOceanException(int errorCode,
                              String errorDesc)
}
```

Constructor for EnOceanException
Class EnOceanException

Parameters:
errorCode - An error code.

EnOceanException

public EnOceanException(int errorCode)

Constructor for EnOceanException

Parameters:
errorCode - An error code.

Method Detail

errorCode

public int errorCode()

Constructor for EnOceanException

Returns:
An EnOcean error code, defined by the EnOcean Forum working committee or an EnOcean vendor.
public interface EnOceanHandler

The interface used to get callback answers from a RPC or a Message.

Version:

1.0

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><code>notifyResponse(EnOceanRPC original, byte[] payload)</code></td>
</tr>
<tr>
<td></td>
<td>Notifies of the answer to a RPC.</td>
</tr>
</tbody>
</table>

Method Detail

`notifyResponse`

void `notifyResponse(EnOceanRPC original, byte[] payload)`

Notifies of the answer to a RPC.

Parameters:

- `original` - the original `EnOceanRPC` that originated this answer.
- `payload` - the payload of the response; may be deserialized to an `EnOceanRPC` object.
public interface EnOceanHost

This interface represents an EnOcean Host, a device that offers EnOcean networking features.

Version:
1.0

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST_ID</td>
<td>45</td>
</tr>
</tbody>
</table>

- **HOST_ID**: The unique ID for this Host: this matches the CHIP_ID of the EnOcean Gateway Chip it embodies.

REPEATER_LEVEL_OFF

- **REPEATER_LEVEL_OFF**: repeater level to disable repeating; this is the default.

REPEATER_LEVEL_ONE

- **REPEATER_LEVEL_ONE**: repeater level to repeat every telegram at most once.

REPEATER_LEVEL_TWO

- **REPEATER_LEVEL_TWO**: repeater level to repeat every telegram at most twice.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiVersion()</td>
<td>46</td>
</tr>
<tr>
<td>appVersion()</td>
<td>46</td>
</tr>
<tr>
<td>getBaseID()</td>
<td>47</td>
</tr>
<tr>
<td>getChipId(String servicePID)</td>
<td>47</td>
</tr>
<tr>
<td>getRepeaterLevel()</td>
<td>47</td>
</tr>
<tr>
<td>reset()</td>
<td>46</td>
</tr>
<tr>
<td>setBaseID(int baseID)</td>
<td>47</td>
</tr>
<tr>
<td>setRepeaterLevel(int level)</td>
<td>47</td>
</tr>
</tbody>
</table>

Field Detail

**HOST_ID**

- **HOST_ID**: public static final Object HOST_ID

  The unique ID for this Host: this matches the CHIP_ID of the EnOcean Gateway Chip it embodies.
Interface EnOceanHost

REPEATER_LEVEL_OFF

```java
public static final int REPEATER_LEVEL_OFF = 0
```

repeater level to disable repeating; this is the default.

---

REPEATER_LEVEL_ONE

```java
public static final int REPEATER_LEVEL_ONE = 1
```

repeater level to repeat every telegram at most once.

---

REPEATER_LEVEL_TWO

```java
public static final int REPEATER_LEVEL_TWO = 2
```

repeater level to repeat every telegram at most twice.

### Method Detail

**reset**

```java
void reset()
throws EnOceanException
```

Reset the EnOcean Host (cf. ESP3 command 0x02: C0_WR_RESET)

**appVersion**

```java
String appVersion()
throws EnOceanException
```

Returns the chip's application version info (cf. ESP3 command 0x03: C0_RD_VERSION)

**apiVersion**

```java
String apiVersion()
throws EnOceanException
```

Returns the chip's API version info (cf. ESP3 command 0x03: C0_RD_VERSION)
getBaseID

```java
int getBaseID() throws EnOceanException

Gets the BASE_ID of the chip, if set (cf. ESP3 command 0x08: C0_RD_IDBASE)

Returns:
the BASE_ID of the device as defined in EnOcean specification
Throws:
EnOceanException
```

setBaseID

```java
void setBaseID(int baseID) throws EnOceanException

Sets the base ID of the device, may be used up to 10 times (cf. ESP3 command 0x07:
C0_WR_IDBASE)

Parameters:
baseID - to be set.
Throws:
EnOceanException
```

setRepeaterLevel

```java
void setRepeaterLevel(int level) throws EnOceanException

Sets the repeater level on the host (cf. ESP3 command 0x09: C0_WR_REPEATER)

Parameters:
level - one of the Repeater Level constants as defined above.
Throws:
EnOceanException
```

getRepeaterLevel

```java
int getRepeaterLevel() throws EnOceanException

Gets the current repeater level of the host (cf. ESP3 command 0x0A: C0_RD_REPEATER)

Returns:
one of the Repeater Level constants as defined above.
Throws:
EnOceanException
```

getchipId

```java
int getChipId(String servicePID) throws EnOceanException
```
Retrieves the CHIP_ID associated with the given servicePID, if existing on this chip.

Returns:
the associated CHIP_ID of the exported device.

Throws:  
EnOceanException
### Interface EnOceanMessage

**org.osgi.service.enocean**

```java
public interface EnOceanMessage
```

Holds the necessary methods to interact with an EnOcean message.

**Version:**

1.0

---

#### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getBytes()</code></td>
<td>Gets the byte[], including the CRC.</td>
<td>50</td>
</tr>
<tr>
<td><code>getDbm()</code></td>
<td>Returns the average RSSI on all the received subtelegrams, including redundant ones.</td>
<td>51</td>
</tr>
<tr>
<td><code>getDestinationId()</code></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td><code>getFunc()</code></td>
<td></td>
<td>49</td>
</tr>
<tr>
<td><code>getPayloadBytes()</code></td>
<td>Returns the payload bytes of this message.</td>
<td>50</td>
</tr>
<tr>
<td><code>getRorg()</code></td>
<td></td>
<td>49</td>
</tr>
<tr>
<td><code>getSecurityLevelFormat()</code></td>
<td>Returns the security level of this message, as specified in the 'Security of EnOcean Radio Networks' draft, section 4.2.3.</td>
<td>51</td>
</tr>
<tr>
<td><code>getSenderId()</code></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td><code>getStatus()</code></td>
<td>Gets the current EnOcean status of the Message.</td>
<td>50</td>
</tr>
<tr>
<td><code>getSubTelNum()</code></td>
<td>Returns the number of subtelegrams (usually 1) this Message carries.</td>
<td>51</td>
</tr>
<tr>
<td><code>getType()</code></td>
<td></td>
<td>50</td>
</tr>
</tbody>
</table>

---

#### Method Detail

**getRorg**

```java
int getRorg()
```

Returns:

- the message's RORG

**getFunc**

```java
int getFunc()
```

Returns:

- the message's FUNC
Interface EnOceanMessage

getTypetype

int getType()

Returns:
the message's TYPE

getSenderId

int getSenderId()

Returns:
the message's Sender ID

getDestinationId

int getDestinationId()

Returns:
the message's destination ID, or -1

getBytes

byte[] getBytes()

Gets the bytes corresponding to the whole message, including the CRC. The generated byte[] array may be sent to an EnOcean gateway and is conform to EnOcean Radio Protocol.

Returns:
The serialized byte list corresponding to the binary message.

getPayloadBytes

byte[] getPayloadBytes()

Returns the payload bytes of this message.

getStatus

int getStatus()

Gets the current EnOcean status of the Message. The 'status' byte is actually a bitfield that mainly holds repeater information, teach-in status, and more or less information depending on the radiotelegram type.

Returns:
the current EnOcean status of this message.
Interface EnOceanMessage

getSubTelNum

int getSubTelNum()

Returns the number of subtelegrams (usually 1) this Message carries.

Returns:
The number of subtelegrams in the case of multiframe messages.

getDbm

int getDbm()

Returns the average RSSI on all the received subtelegrams, including redundant ones.

Returns:
The average RSSI perceived.

getSecurityLevelFormat

int getSecurityLevelFormat()

Returns the security level of this message, as specified in the ‘Security of EnOcean Radio Networks’ draft, section 4.2.3.

Returns:
The security level format.
public interface EnOceanRPC

A very basic interface for RPCs.

Version:
1.0

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION_ID</td>
<td>String</td>
<td>The Function ID property string, used in EventAdmin RPC broadcasting.</td>
</tr>
<tr>
<td>MANUFACTURER_ID</td>
<td>String</td>
<td>The Manufacturer ID property string, used in EventAdmin RPC broadcasting.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getFunctionId()</td>
<td>int</td>
<td>Gets the functionID for this RPC.</td>
</tr>
<tr>
<td>getManufacturerId()</td>
<td>int</td>
<td>Gets the manufacturerID for this RPC.</td>
</tr>
<tr>
<td>getPayload()</td>
<td>byte[]</td>
<td>Gets the current payload of the RPC.</td>
</tr>
<tr>
<td>getSenderId()</td>
<td>int</td>
<td>Sets the RPC's senderID.</td>
</tr>
<tr>
<td>setPayload(byte[] data)</td>
<td>void</td>
<td>Sets the current payload of the RPC.</td>
</tr>
<tr>
<td>setSenderId(int chipId)</td>
<td>void</td>
<td>Sets the RPC's senderID.</td>
</tr>
</tbody>
</table>

Field Detail

MANUFACTURER_ID

public static final String MANUFACTURER_ID = "enocean.rpc.manufacturer_id"

The Manufacturer ID property string, used in EventAdmin RPC broadcasting.

FUNCTION_ID

public static final String FUNCTION_ID = "enocean.rpc.function_id"

The Function ID property string, used in EventAdmin RPC broadcasting.
Interface EnOceanRPC

Method Detail

getManufacturerId

int getManufacturerId()

   Gets the manufacturerID for this RPC.

getFunctionId

int getFunctionId()

   Gets the functionID for this RPC.

getPayload

byte[] getPayload()

   Gets the current payload of the RPC.

   Returns:
   the payload, in bytes, of this RPC.

setPayload

void setPayload(byte[] data)

   Sets the current payload of the RPC.

   Parameters:
   data - the payload, in bytes, of this RPC.

getSenderId

int getSenderId()

   Sets the RPC's senderID. This member has to belong to EnOceanRPC interface, for the object may be sent as a standalone using EventAdmin for instance.

setSenderId

void setSenderId(int chipId)

   Sets the RPC's senderID.
public interface EnOceanSerialInOut

This class is intended to provide in/out methods from serial to emulate an EnOcean dongle during integration tests.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getInputStream()</td>
<td>Returns a handle to the current input stream used by the driver's host.</td>
<td>54</td>
</tr>
<tr>
<td>getOutputStream()</td>
<td>Returns a handle to the current output stream used by the driver's host.</td>
<td>54</td>
</tr>
<tr>
<td>resetBuffers()</td>
<td>Resets the data being exchanged on both input and output streams.</td>
<td>54</td>
</tr>
</tbody>
</table>

### Method Detail

**resetBuffers**

```java
void resetBuffers()
```

Resets the data being exchanged on both input and output streams.

**getInputStream**

```java
InputStream getInputStream()
```

Returns a handle to the current input stream used by the driver's host.

**getOutputStream**

```java
OutputStream getOutputStream()
```

Returns a handle to the current output stream used by the driver's host.
## Package org.osgi.service.enocean.descriptions

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EnOceanChannelDescription</code></td>
<td>56</td>
</tr>
<tr>
<td>Public and registered description interface for a channel.</td>
<td></td>
</tr>
<tr>
<td><code>EnOceanChannelDescriptionSet</code></td>
<td>59</td>
</tr>
<tr>
<td>This interface represents an EnOcean Channel Description Set.</td>
<td></td>
</tr>
<tr>
<td><code>EnOceanChannelEnumValue</code></td>
<td>60</td>
</tr>
<tr>
<td>This transitional interface is used to define all the possible values taken by an enumerated channel.</td>
<td></td>
</tr>
<tr>
<td><code>EnOceanDataChannelDescription</code></td>
<td>61</td>
</tr>
<tr>
<td>Subinterface of <code>EnOceanChannelDescription</code> that describes physical measuring channels.</td>
<td></td>
</tr>
<tr>
<td><code>EnOceanEnumChannelDescription</code></td>
<td>63</td>
</tr>
<tr>
<td>Subinterface of <code>EnOceanChannelDescription</code> that describes enumerated channels.</td>
<td></td>
</tr>
<tr>
<td><code>EnOceanFlagChannelDescription</code></td>
<td>64</td>
</tr>
<tr>
<td>Subinterface of <code>EnOceanChannelDescription</code> that describes boolean channels.</td>
<td></td>
</tr>
<tr>
<td><code>EnOceanMessageDescription</code></td>
<td>65</td>
</tr>
<tr>
<td><code>EnOceanMessageDescriptionSet</code></td>
<td>66</td>
</tr>
<tr>
<td>This interface represents an EnOcean Message Description Set.</td>
<td></td>
</tr>
<tr>
<td><code>EnOceanRPCDescription</code></td>
<td>67</td>
</tr>
<tr>
<td><code>EnOceanRPCDescriptionSet</code></td>
<td>68</td>
</tr>
<tr>
<td>This interface represents an EnOcean RPC Set.</td>
<td></td>
</tr>
</tbody>
</table>
Interface EnOceanChannelDescription

All Known Subinterfaces:
- EnOceanDataChannelDescription
- EnOceanEnumChannelDescription
- EnOceanFlagChannelDescription

```java
public interface EnOceanChannelDescription {
    public interface EnOceanChannelDescription

    Public and registered description interface for a channel. Encompasses all the possible subtypes for a channel.

    Version:
    1.0
```

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANNEL_ID</td>
<td>The unique ID of this EnOceanChannelDescription object.</td>
</tr>
<tr>
<td>TYPE_DATA</td>
<td>A DATA channel maps itself to a Double value representing a physical measure.</td>
</tr>
<tr>
<td>TYPE_ENUM</td>
<td>An ENUM channel maps itself to one between a list of discrete</td>
</tr>
<tr>
<td></td>
<td>EnOceanChannelEnumValue &quot;value objects&quot;.</td>
</tr>
<tr>
<td>TYPE_FLAG</td>
<td>A FLAG channel maps itself to a Boolean value.</td>
</tr>
<tr>
<td>TYPE_RAW</td>
<td>A RAW channel is only made of bytes.</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deserialize(byte[])</td>
<td>Tries to deserialize a series of bytes into a documented value object (raw bytes, Double or EnOceanChannelEnumValue).</td>
</tr>
<tr>
<td>getType()</td>
<td>Retrieves the type of the channel.</td>
</tr>
<tr>
<td>serialize(Object obj)</td>
<td>Tries to serialize the channel into a series of bytes.</td>
</tr>
</tbody>
</table>

### Field Detail

**CHANNEL_ID**

```java
public static final String CHANNEL_ID = "enocean.channel.description.channel_id"
```

The unique ID of this EnOceanChannelDescription object.

**TYPE_RAW**

```java
public static final String TYPE_RAW = "enocean.channel.description.raw"
```
Interface EnOceanChannelDescription

A RAW channel is only made of bytes.

### TYPE_DATA

```java
public static final String TYPE_DATA = "enocean.channel.description.data"
```

A DATA channel maps itself to a `Double` value representing a physical measure.

### TYPE_FLAG

```java
public static final String TYPE_FLAG = "enocean.channel.description.flag"
```

A FLAG channel maps itself to a `Boolean` value.

### TYPE_ENUM

```java
public static final String TYPE_ENUM = "enocean.channel.description.enum"
```

An ENUM channel maps itself to one between a list of discrete `EnOceanChannelEnumValue` "value objects".

**Method Detail**

#### getType

```java
String getType()
```

Retrieves the type of the channel.

Returns:

one of the above-described types.

#### serialize

```java
byte[] serialize(Object obj)
```

Tries to serialize the channel into a series of bytes.

Parameters:

- `obj` - the value of the channel.

Returns:

the right-aligned value, in raw bytes, of the channel.

Throws:

- `IllegalArgumentException`

#### deserialize

```java
Object deserialize(byte[] bytes)
```

Throws `IllegalArgumentException`
Tries to deserialize a series of bytes into a documented value object (raw bytes, Double or
EnOceanChannelEnumValue. Of course this method will be specialized for each
EnOceanChannelDescription subinterface, depending on the type of this channel.

Parameters:
  bytes - the right-aligned raw bytes.
Returns:
  a value object.
Throws:
  IllegalArgumentException
Interface EnOceanChannelDescriptionSet

`org.osgi.service.enocean.descriptions`

```
public interface EnOceanChannelDescriptionSet

This interface represents an EnOcean Channel Description Set. `EnOceanChannelDescriptionSet` is
registered as an OSGi Service. Provides a method to retrieve the `EnOceanChannelDescription`
objects it documents.

Version:
1.0
```

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getChannelDescription(String channelId)</code></td>
<td>59</td>
</tr>
</tbody>
</table>

### Method Detail

**getChannelDescription**

```java
EnOceanChannelDescription getChannelDescription(String channelId)
```

Retrieves a `EnOceanChannelDescription` object according to its identifier.

**Returns:**

The corresponding `EnOceanChannelDescription` object, or null.

**Throws:**

`IllegalArgumentException` - if the supplied String is invalid, null, or other reason.
public interface EnOceanChannelEnumValue

This transitional interface is used to define all the possible values taken by an enumerated channel.

Version:
1.0

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td><strong>getDescription()</strong></td>
</tr>
<tr>
<td></td>
<td>A non-mandatory description of what this enumerated value is about.</td>
</tr>
<tr>
<td>int</td>
<td><strong>getStart()</strong></td>
</tr>
<tr>
<td></td>
<td>The start value of the enumeration.</td>
</tr>
<tr>
<td>int</td>
<td><strong>getStop()</strong></td>
</tr>
<tr>
<td></td>
<td>The stop value of the enumeration.</td>
</tr>
</tbody>
</table>

Method Detail

**getStart**

```java
int getStart()
```

The start value of the enumeration.

Returns:
the start value.

**getStop**

```java
int getStop()
```

The stop value of the enumeration.

Returns:
the stop value.

**getDescription**

```java
String getDescription()
```

A non-mandatory description of what this enumerated value is about.

Returns:
the english description of this channel.
Interface EnOceanDataChannelDescription

org.osgi.service.enocean.descriptions

All Superinterfaces:
   EnOceanChannelDescription

public interface EnOceanDataChannelDescription extends EnOceanChannelDescription

Subinterface of EnOceanChannelDescription that describes physical measuring channels.

Version: 1.0

Fields inherited from interface org.osgi.service.enocean.descriptions.EnOceanChannelDescription
CHANNEL_ID, TYPE_DATA, TYPE_ENUM, TYPE_FLAG, TYPE_RAW

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDomainStart()</td>
<td>61</td>
</tr>
<tr>
<td>getDomainStop()</td>
<td>61</td>
</tr>
<tr>
<td>getRangeStart()</td>
<td>61</td>
</tr>
<tr>
<td>getRangeStop()</td>
<td>62</td>
</tr>
<tr>
<td>getUnit()</td>
<td>62</td>
</tr>
</tbody>
</table>

Methods inherited from interface org.osgi.service.enocean.descriptions.EnOceanChannelDescription
deserialize, getType, serialize

Method Detail

getDomainStart

int getDomainStart()

Returns:
   the start of the raw input range for this channel.

g DOMAINSTOP

int getDomainStop()

Returns:
   the end of the raw input range for this channel.

g RANGESTART

double getRangeStart()

...
Interface EnOceanDataChannelDescription

Returns:
the scale start at which this channel will be mapped to (-20,0°C for instance)

getRangeStop

double getRangeStop()

Returns:
the scale stop at which this channel will be mapped to (+30,0°C for instance)

getUnit

String getUnit()

Returns:
the non-mandatory physical unit description of this channel.
Interface EnOceanEnumChannelDescription

org.osgi.service.enocean.descriptions

All Superinterfaces:
   EnOceanChannelDescription

public interface EnOceanEnumChannelDescription extends EnOceanChannelDescription

Subinterface of EnOceanChannelDescription that describes enumerated channels.

Version: 1.0

Fields inherited from interface org.osgi.service.enocean.descriptions.EnOceanChannelDescription

CHANNEL_ID, TYPE_DATA, TYPE_ENUM, TYPE_FLAG, TYPE_RAW

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnOceanChannelEnumValue[] getPossibleValues()</td>
<td>63</td>
</tr>
</tbody>
</table>

Methods inherited from interface org.osgi.service.enocean.descriptions.EnOceanChannelDescription
deserialize, getType, serialize

Method Detail

getPossibleValues

EnOceanChannelEnumValue[] getPossibleValues()

Returns:
    all the possible values for this channel.
Interface EnOceanFlagChannelDescription

org.osgi.service.enocean.descriptions

All Superinterfaces:
   EnOceanChannelDescription

public interface EnOceanFlagChannelDescription
extends EnOceanChannelDescription

Subinterface of EnOceanChannelDescription that describes boolean channels.

Version:
   1.0

Fields inherited from interface org.osgi.service.enocean.descriptions.EnOceanChannelDescription
CHANNEL_ID, TYPE_DATA, TYPE_ENUM, TYPE_FLAG, TYPE_RAW

Methods inherited from interface org.osgi.service.enocean.descriptions.EnOceanChannelDescription
deserialize, getType, serialize
public interface EnOceanMessageDescription

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deserialize(byte[] bytes)</td>
<td>Deserializes an array of bytes into the EnOceanChannels available to the payload, if possible.</td>
</tr>
<tr>
<td>serialize(EnOceanChannel[] channels)</td>
<td>Serializes a series of EnOceanChannel objects into the corresponding byte[] sequence.</td>
</tr>
</tbody>
</table>

Method Detail

serialize

byte[] serialize(EnOceanChannel[] channels)
throws EnOceanException

Serializes a series of EnOceanChannel objects into the corresponding byte[] sequence.

Throws:
EnOceanException

deserialize

EnOceanChannel[] deserialize(byte[] bytes)
throws EnOceanException

Deserializes an array of bytes into the EnOceanChannels available to the payload, if possible. If the actual instance type of the message is not compatible with the bytes it is fed with (RORG to begin with), throw an IllegalArgumentException.

Throws:
EnOceanException
public interface EnOceanMessageDescriptionSet

This interface represents an EnOcean Message Description Set. EnOceanMessageDescriptionSet is registered as an OSGi Service. Provides method to retrieve the EnOceanMessageDescription objects it documents.

Version:
1.0

Method Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnOceanMessageDescription.getMessageDescription</td>
<td>Retrieves an EnOceanMessageDescription object according to its identifiers.</td>
</tr>
</tbody>
</table>

Method Detail

getMessageDescription

EnOceanMessageDescription getMessageDescription(int rorg, int func, int type, int extra)

Retrieves a EnOceanMessageDescription object according to its identifiers. See EnOcean Equipment Profile Specification for more details.

Parameters:
- rorg - the radio telegram type of the message.
- func - The func subtype of this message.
- type - The type subselector.
- extra - Some extra information; some EnOceanMessageDescription objects need an additional specifier. If not needed, has to be set to -1.

Returns:
The EnOceanMessageDescription object looked for, or null.

Throws:
- IllegalArgumentException - if there was an error related to the input arguments.
public interface EnOceanRPCDescription

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getName()</td>
<td>Get a friendly name for the RPC</td>
<td>67</td>
</tr>
</tbody>
</table>

Method Detail

g>Name

String getName()

Get a friendly name for the RPC
Interface EnOceanRPCDescriptionSet

org.osgi.service.enocean.descriptions

public interface EnOceanRPCDescriptionSet

This interface represents an EnOcean RPC Set. EnOceanRPCDescriptionSet is registered as an OSGi Service. Provides a method to retrieve the EnOceanRPC objects it documents.

Version:
1.0

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnOceanRPCDescription.getRPC(short manufacturerId, short commandID)</td>
<td>68</td>
</tr>
</tbody>
</table>

Retrieves a EnOceanRPC object according to its identifier.

Method Detail

getRPC

EnOceanRPCDescription getRPC(short manufacturerId, short commandID)

throws IllegalArgumentException

Retrieves a EnOceanRPC object according to its identifier.

Returns:
The corresponding EnOceanRPC object, or null.

Throws:
IllegalArgumentException

Java API documentation generated with DocFlex/Doclet v1.5.6

DocFlex/Doclet is both a multi-format Javadoc doclet and a free edition of DocFlex/Javadoc. If you need to customize your Javadoc without writing a full-blown doclet from scratch, DocFlex/Javadoc may be the only tool able to help you! Find out more at www.docflex.com

8  Considered Alternatives

June 19, 2013:
The RFC’s style has been changed to be more lightly descriptive on the actual Java types. The implementation efforts in parallel, on the opposite, are driving the actual interface specification that is now put and update in the "Java Interface Specification" paragraph above.

About the dynamic implementation of Messages, as proposed by N. Portinaro, it is decided to keep using interfaces only in the specification, and not define classes. Nevertheless, a sample of such a dynamic implementation using anonymous classes implementing those interfaces on-the-fly has been
proposed as an example

June 4, 2013:
It has been discussed whether or not to add a setChannels() or appendChannels() method to the
EnOceanMessage interface. Unfortunately, trying to do so resulted in a cluttered and difficult to read
interface for no clear benefits, so it has been decided not to add it yet.

The possibility to send messages using the EnOceanHost interface has also been declined yet; this
interface should be used for the configuration of the gateway chip only.

After an evaluation of the issues and rewards that would bring an implementation of the scarcely-used
SmartAck protocol, it has been decided to set it aside for this iteration.

May 27, 2013:
The 'export' feature is not anymore set aside and should be challenged for consideration. The
'SmartAck' feature status is still under evaluation. The Security features of EnOcean have been
integrated. Remote Management is integrated in a minimal fashion. Since there is no clear specification
on the equivalent of the 'datatypes' for Remote Management, it has been decided yet (V. Perron) to set
aside the development of abstractions for them and let the programmer implement extra methods
above the specification when deemed useful.

April 03, 2013:
It has been decided (A. Bottaro, M. Robin, V. Perron) to set aside the 'export' feature of EnOcean
device service for further reflexion, as well as 'SmartAck' feature from EnOcean, which will require extra
effort. For this latter topic, one of the tracks worth exploring was the setup of Mailboxes at the
EnOceanHost level (which sticks to the reality of the EnOcean gateway chips) and recommending a
dedicated 'real-time' channel to be implemented, so that SmartAck message frames could be carried
synchronously.

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 Annex

EnOcean Networking
EnOcean networking is a quite particular wireless network in the sense that there is no actual “topology”. Every
device emits messages on the same frequency band, which depends on the world region and local regulations.
In Europe, the 868 MHz frequency band is used; in Asia, the 315 MHz is adopted. The 902 MHz band is in the
process of being used for North America. There is no notion of a “network identifier” in EnOcean.
The transmitting devices usually broadcast all of their messages on this frequency, and most of the time do not
wait for an answer. The transmitting devices being mostly energy-harvesting devices, they cannot easily wait for
an answer.
The receiver modules listen to every message sent on the frequency band. They filter the messages of interest based on the Sender ID that is embedded within every message. They are supposed to listen only to Sender ID that have previously been “taught” to them, and discard the others.

The teach-in procedure is specific to EnOcean. The receiver module has to be manually (or remotely, but that is still very rare) switched to a “learning mode”. It will wait for a special kind of EnOcean messages, called “teach-in” messages. Those “teach-in” messages have to be sent by the emitting device that is targeted to be learnt by the receiver.

Once the receiver module has received this “teach-in” message, it should keep in non-volatile memory the sender ID of that message and such, be “paired” with it.

Because of this process, EnOcean networks are N-to-N: you may pair N emitters to 1 receiver, 1 emitter with N receivers, or even N emitters to M receivers.

In this respect, the EnOcean gateway is somewhat special; it is a device able to both send and receive messages, is line-powered, and listens to every message in the frequency band.

**EnOcean Network Security**

The security in EnOcean exists in a point-to-point fashion. The emitting device will be responsible of transmitting the optional Key and/or Rolling Authentication Code (RLC) to the receiver device during a dedicated “Security-Teach-In” phase.

The security configuration, Key and RLC are transmitted using a special message. The receiver device then associates the given key and RLC to that device internally, and uses them to decode any further message coming from it.

As a result, the Key and RLC parameters, as well as the current security configuration, are properties tied to a sending EnOceanDevice object; furthermore, an arbitrary number of security configurations, keys and RLCs may coexist within the same EnOcean network.

*It will be the responsibility of the receiver object to fetch the current security properties of the sending object and use them to decode further messages.*

### 11 Document Support

**References**


[5]. EnOcean Equipment Profiles v2.5, EnOcean Alliance, March 04, 2013


[7]. EnOcean System Specification – Smart Acknowledgment v1.4, EnOcean Alliance, September 15, 2010


### Author's Address

<table>
<thead>
<tr>
<th>Name</th>
<th>André BOTTARO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>France Telecom Orange</td>
</tr>
<tr>
<td>Address</td>
<td>28 Chemin du Vieux Chêne, Meylan, France</td>
</tr>
<tr>
<td>Voice</td>
<td>+33 4 76 76 41 03</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:andre.bottaro@orange.com">andre.bottaro@orange.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Maïlys ROBIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Orange Labs Tokyo</td>
</tr>
<tr>
<td>Address</td>
<td>Keio Shinjuku Oiwake Bldg. 9F, 3-1-13 Shinjuku, Shinjuku-ku, Tokyo 160-0022</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:mrobin.ext@orange.com">mrobin.ext@orange.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Victor PERRON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Orange Labs Tokyo</td>
</tr>
<tr>
<td>Address</td>
<td>230 rue La Fayette, 75010 PARIS</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:victor.perron@orange.fr">victor.perron@orange.fr</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Antonin CHAZALET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Orange</td>
</tr>
<tr>
<td>Address</td>
<td>28 Chemin du Vieux Chêne, Meylan, France</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:antonin.chazalet@orange.com">antonin.chazalet@orange.com</a></td>
</tr>
</tbody>
</table>

### Acronyms and Abbreviations

### End of Document
Abstract

OSGi applications need hardware and software resources to perform their features. As these resources are limited, they have to be fairly shared between applications in order to preserve the global quality of service. Up to now, OSGi platforms delegate the resource management features to the Java Virtual Machines themselves or took advantage of some external Java Resource Management solution like JVM TI or JMX. Unfortunately, all of these resource management solutions provide features at the Object or Class level. This granularity is too low level to easily monitor resources consumed by OSGi applications.

This specification proposes a Resource Management solution fitting with OSGi model and constraints. Resources (CPU, memory, disk storage space, I/O) are monitored per bundle and can be enabled and disabled at runtime. When a bundle consumes too many resources, the Resource Management solution notifies interested applications and limits resource allocations.
0 Document Information

License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGI ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, 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.

Trademarks

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

Feedback

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

Table of Contents

0 Document Information..................................................................................................2
   License..................................................................................................................2
   Trademarks..........................................................................................................3
   Feedback..............................................................................................................3
   Table of Contents...............................................................................................3
   Terminology and Document Conventions.................................................................4
   Revision History....................................................................................................5

1 Introduction.............................................................................................................9

2 Application Domain...............................................................................................10
   What are resources?.................................................................................................10
   Most common and crucial resources.....................................................................11
   What is Healthiness?...............................................................................................12
   Terminology and abbreviations...........................................................................12

3 Problem Description...............................................................................................13
   Cooperative applications.......................................................................................13
   Less cooperative or legacy applications.................................................................14

4 Requirements.........................................................................................................14

5 Initial Specification Chapter..................................................................................15
   Essentials...............................................................................................................15
   Entities....................................................................................................................16
   Operation summary...............................................................................................17
Resource Management

Draft

August 1, 2014

Resource Context ................................................................. 18
System Resource Context .................................................. 18
Framework Resource Context ............................................. 18
Resource Monitor .............................................................. 18
Resource Monitor Factory ............................................... 19
CPU Monitor ................................................................. 20
Memory Monitor .............................................................. 20
Socket Monitor ............................................................... 20
Disk Storage Monitor ....................................................... 20
Thread Monitor ............................................................... 20
Resource Listener .......................................................... 21
Resource Event .............................................................. 24
Resource Context Listener ................................................ 24
Resource Context Event .................................................... 25
Resource Manager .......................................................... 25
Resource Management Authority ..................................... 26

6 Javadoc .................................................................................. 26

7 Considered Alternatives ..................................................... 66
   Resource Manager inside the Core framework or in a bundle? ............................................. 66
   Adapt pattern or OSGi service? ......................................................................................... 66
   Eventing paradigms ......................................................................................................... 66
   Resource event classes .................................................................................................... 66
   CPU Monitor ................................................................................................................ 67
   Resource Threshold algorithm and eventing ................................................................. 67
   Resource Monitor Factory ............................................................................................... 68
   Resource accounting and context switching .................................................................. 68
   Compatibility between bundles capabilities ................................................................ 72
      Implementation of resource management aware bundles ............................................ 72
      Example 1 .................................................................................................................... 73
      Example 2 .................................................................................................................... 73
      Event Admin use-case ................................................................................................. 74
      Http Service use-case ................................................................................................. 75

8 Security Considerations ....................................................... 76

9 Document Support ............................................................... 76
   References ...................................................................................................................... 76
   Author’s Address ............................................................................................................ 77
   Acronyms and Abbreviations ......................................................................................... 78
   End of Document ............................................................................................................ 78

Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.
## 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>
</table>
| Initial  | 03 06 2013 | Initial draft.  
RINQUIN Arnaud, Orange, arnaud.rinquin@orange.com  
BOTTARO André, Orange, andre.bottaro@orange.com  
BONNARDEL Grégory, Orange, gbonnardel.ext@orange.com |
| v01      | 04 16 2013 | Some modifications after the Cologne meeting:  
- Global Resource Context is now named as Platform Resource Context  
- Bandwidth Monitor → Socket Monitor  
- Resource Context persistence  
- Add a table indicating the type of Java object the ResourceMonitor.getUsage() method returns for each type of resource.  
- New paragraph about the use of the context switching operation (chapter 6.15 Resource Manager) |
| V02      | 04 30 2013 | Fixes:  
- Rewritings  
  - Operation Summary  
  - CPU Monitor  
  - Socket Monitor  
  - Framework Resource Context  
- Actions are taken by authorities  
Added:  
- Resource Thresholds  
- Resource Event Types  
  - Back To Normal  
  - Back To Warning |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>V03</td>
<td>05 14 2013</td>
<td>Fixes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Rewritings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ Memory Monitor (OutOfMemoryException)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ Implementation of all types of Resource Monitor is optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ Socket Monitor (existing sockets)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ BACK_TO_NORMAL Resource Event → NORMAL Resource Event</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ BACK_TO_WARNING Resource Event → WARNING Resource Event</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A new paragraph about algorithms reducing the number of Resource Event into the Considered Alternatives chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Socket Monitor scope in Considered Alternatives</td>
</tr>
<tr>
<td>V04</td>
<td>05 22 2013</td>
<td>Fixes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Socket Monitor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ only tracks bound or connected socket</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ throw a SocketException when an error threshold is reached</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Rewritings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ Disk Storage Monitor (exception)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ Thread Monitor (exception)</td>
</tr>
<tr>
<td>V05</td>
<td>05 28 2013</td>
<td>Modifications:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Add Resource Monitor Factory (new paragraph + entity diagram)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Change resource monitor threshold list to maximum and minimum threshold attributes</td>
</tr>
<tr>
<td>Revision</td>
<td>Date</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| V06      | 06 04 2013 | Modifications:  
  - Clarify the Resource Monitor state when a new instance is created by a ResourceMonitorFactory.  
  - ResourceEvent.getResourceThreshold() returns a SNAPSHOT of the Resource Threshold instance at the moment when the event was generated.  
  - Resource Thresholds:  
    ◦ A Resource Monitor holds an upper Resource Threshold instance and a lower Resource Threshold.  
    ◦ Add a diagram showing Resource Threshold state transitions. This diagram also shows what kind of Resource Event is generated.  
  - Clarify which threads are monitored by a ThreadMonitor (alive thread = all threads which are in the RUNNABLE, BLOCKED, WAITING, TIMED_WAITING java state). |
| V07      | 06 12 2013 | Modifications:  
  - Refactoring of the Resource Event interface:  
    ◦ Add getResourceType() method  
    ◦ Add getValue() method  
    ◦ Add getThresholdValue() method  
    ◦ Add isUpperThreshold() method  
    ◦ Remove getMonitor() method  
    ◦ Remove getThreshold() method  
    ◦ Remove getMonitor() method |
| V08      | 06 19 2013 | Updates:  
  - Replace ResourceMonitor.setMonitored(boolean) by ResourceMonitor.enable()/disable()  
  - Replace ResourceMonitor.isMonitored() by ResourceMonitor.isEnabled()  
  - Threshold diagram (new colors + state) |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>V09</td>
<td>06 26 2013</td>
<td>Updates:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Resource Monitor Factory chapter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ the newly created Resource Monitor instance is disabled by default because it should be configured before activation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ Resource Monitor instance MUST be created only by Resource Monitor Factory instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Add ResourceMonitor,isDeleted() method.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Considered Alternatives</td>
</tr>
<tr>
<td>V10</td>
<td>07 04 2013</td>
<td>Updates:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Add a new diagram for upper threshold (cpu example)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The upper and lower threshold diagram has been adapted to the socket resource</td>
</tr>
<tr>
<td>V11</td>
<td>07 10 2013</td>
<td>• Replaced arrows by circles into diagrams related to Resource Thresholds. Removed the text on arrows/circle. Update diagram titles.</td>
</tr>
<tr>
<td>V12</td>
<td>07 18 2013</td>
<td>• Remove old references to connected socket → in-use state sockets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Remove old references to active/started threads → alive threads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Remove old reference to Bandwidth Monitor → Socket Monitor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Update Extensibility clause of the Essentials chapter (5 types of resources instead of 4).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New use case for the context switching operation (Event Handler use case)</td>
</tr>
<tr>
<td>V13</td>
<td>07 23 2013</td>
<td>• A Socket Monitor tracks native socket file descriptors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Update UML Schema:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ add Resource Listener Implementer entity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ add Resource Monitor Factory Implementer entity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Update Operation Summary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Add new section in Considered Alternatives about compatibility implementation between bundles which handles ResourceManagement api and the other ones.</td>
</tr>
<tr>
<td>V14</td>
<td>07 30 2013</td>
<td>• Add EventAdmin and HttpService use cases into compatibility section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Describe how a bundle can manage Resource Management features (direct implementation, weaving, service proxy)</td>
</tr>
<tr>
<td>Revision</td>
<td>Date</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>V15</td>
<td>08 06 2013</td>
<td>• ResourceManager entity registered as a service and accessible through the adapt method.</td>
</tr>
<tr>
<td>V16</td>
<td>08 14 2013</td>
<td>• Introduce Resource Context Listener and Resource Context Event</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• All Resource Context are retrieved through the Resource Manager service (Bundle.adapt() approach is deprecated)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Thresholds are now hosted by Resource Listener</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Resource Threshold entities have been removed.</td>
</tr>
<tr>
<td>V17</td>
<td>09 26 2013</td>
<td>• Exceptions thrown when a threshold is reached are now optional.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Use of MemoryException instead of OutOfMemoryException</td>
</tr>
<tr>
<td>V18</td>
<td>11 06 2013</td>
<td>• Introduce new methods in ResourceMonitor reporting resource context operations.</td>
</tr>
<tr>
<td>V19</td>
<td>July 24, 2014</td>
<td>Antonin CHAZALET, Orange</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Fix typos.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Add several comments.</td>
</tr>
<tr>
<td>v20</td>
<td>July 25th, 2014</td>
<td>André Bottaro, Orange</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Removed context switching feature in the specification chapter part. Put the main context switching feature descriptions into ‘considered alternatives’ section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Partially removed the link between resource context and threads. To be continued.</td>
</tr>
<tr>
<td>V21</td>
<td>July 30th, 2014</td>
<td>Antonin CHAZALET, Orange</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Add javadoc.</td>
</tr>
<tr>
<td>V22</td>
<td>August 1st, 2014</td>
<td>Antonin CHAZALET, Orange, André BOTTARO, Orange</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Remove “links” amongst Resource Context, and Thread.</td>
</tr>
</tbody>
</table>
1 Introduction

Applications, executed on an OSGi platform, need hardware resources (CPU, memory, disk, storage space) and software resources (sockets, threads). As these resources are limited, applications have to share them in order to preserve system quality of service. This is a general fact in the Enterprise and Residential markets.

Providing fair resource management features is crucial for the Smart Home to emerge as Residential players are opening their gateway (or box) execution environment to third party applications. In this perspective, the framework administrator has to fairly offer the same guarantees to every actor sharing the platform.

Resource Monitoring is also vital to Cloud Computing scenarios where a management agent needs to ensure that SLAs agreed around the cloud offering are met. When a cloud node gets overloaded or fails this can affect the pre-agreed SLA and action needs to be taken. In a Cloud Computing scenario this may imply starting additional nodes, adjusting the provisioning state of the system by moving or adding deployments or indeed shutting down some nodes if the system become quiet. To be able to handle such scenarios the management agent will need to have visibility of the resource utilization of the cloud system as a whole, which encompasses a multiplicity of nodes and runtimes.

For the moment, existing OSGi specifications do not provide monitoring resource mechanism ensuring a fair resource sharing between bundles and applications. The underlying JVM provides only some standard mechanisms at a level that is too fine-grained, e.g., classes, objects, methods. The bundle being the smallest deployment unit of interest for platform administrator and application provides, this RFC defines an API compliant with the RFP [3].

Introduce the RFC. Discuss the origins and status of the RFC and list any open items to do.

2 Application Domain

Resources of environments are always limited and entities that share such environments should be aware of that. This is not different in OSGi environments. Each bundle consumes resources of different types. Some of them are required for the very basic operations, some others are nice to have, but all of them can run out and lead to situations where the bundle, a set of bundles that form an application, or even the framework as a whole is not operational anymore.

Problematic situations arise when a software unit binds a lot of resources but does not release them after normal operation. This can be caused by wrong implementations, wrong error handling or by intention in case of malware. Especially in environments with very limited resources and/or with a huge number of bundles/vendors it is crucial to monitor the state of bundles and their resource consumption and also to provide mechanisms to react on detected failures.

What are resources?

There are some obvious, basic resources like CPU, memory, disk-space, bandwidth. But new applications might introduce the need for new, different types of resources that are required for their normal operation (e.g., the
presence of certain external services and devices, room temperature etc.). Because of that it is impossible to provide a complete list of potential resources here. The following figure tries to illustrate that:

![Diagram showing origins of resources]

**Figure 1: Origins of resources**

Every circle in this picture stands for a certain resource. As illustrated these resources can come from:

- the same OSGi Framework (e.g., service instances, exported packages …),
- the same Java VM (e.g., threads, memory …),
- the device (e.g., USB-Ports, network interfaces/ports …),
- the local environment (e.g., room temperature, power consumption of the device, geo-location…),
- or from completely external locations (e.g., special external services like maps, dictionary …).

### Most common and crucial resources

Applications use hardware and operating system resources. Targeted resources are:

- CPU
- Memory
- Disk storage space
- Bandwidth on connected networks

JVMs allocate these resources when applications call Java standard APIs. They may provide resource monitoring mechanisms such as:
• Java Management Extension (JMX), now provided by all J2SE-v5-compliant JVM
• JVM Tool Interface Interface(JVMTI) and JVM Profiler Interface (JVMPI)
• Proprietary resource management API (e.g., IBM J9, Oracle Java Embedded Client, /K/ Embedded Mika Max, Myriad Jbed)

The latter provide strict algorithms that charge bundles with consumed resources. There are two known algorithms [4]:

• Direct accounting: the resources consumed during bundle interaction are accounted to the code provider. In other words, the CPU used by a code that belongs to bundle A will be accounted to A, even if it is the bundle B that called this code through a public interface.
• Indirect accounting: all the resources consumed by the threads belonging to a bundle are accounted to this same bundle. Therefore in service interaction there is no resource consumption accounted to service providers.

Java and OSGi enables CPU management per bundle on any VM (without any VM customization) [4],[5]. However, memory management require that standard VMs either implement JVMTI Java standard [8], or implement custom APIs. [6],[7]. These features are not available on standard VMs, with the following definition: a standard Java platform implements the JVM Specification [9], Java Language Specification [10], and base class libraries (http://docs.oracle.com/javase/7/docs/api). It does not include tools like javac, javadoc, JVMTI, tools.jar which are outside the Java platform specification.

What is Healthiness?
Healthiness of an entity (service, bundle, set of bundles, or the whole framework) is meant as the state where the entity is operational as it was specified and will be for the foreseeable future. The correct operation of such an entity is often strongly related to the availability, and perhaps a certain quality, of resources that the entity needs to work. That means an entity that doesn't have or get the required resources is not healthy. There might also be intermediate states where mandatory resources are there, but some optional ones are not available.

Other reasons for non-healthy entities are potential failure situations either inside the entity itself or in their environment. Sometimes such conditions cause shortage of other resources, which at the end affects other entities as well.

So, in order to ensure the healthiness of entities the first step that should be done is to ask themselves, "how do you feel?", and the second step is to know the entities resources requirements, and to monitor their availability.

Terminology and abbreviations

Application
A set of bundles needed to render a full application to the user.

Observable
An entity that is subject of Health monitoring. In the scope of this document this can be a framework, a bundle or a set of bundles.

Health
The state of an observable that describes its ability to work as specified.
Resource
A limited source or supply of physical or virtual goods that are used by bundles in order to provide their service(s).

Fault
The term fault is usually used to name a defect at the lowest level of abstraction, e.g., a memory cell that always returns the value 0.

Error
A fault may cause an error, which is a category of the system state.

Failure
An error, in effect, may lead to a failure, meaning that the system deviates from its correctness specification.

This section should be copied from the appropriate RFP(s). It is repeated here so it can be extended while the RFC authors learn more subtle details.

3 Problem Description

OSGi platforms host several applications which are executed concurrently. These applications have to share limited resources between them.

Cooperative applications
These mechanisms should also allow to estimate the severity of the situation and to decide for required actions to recover the intended state. Ideally, this should be done in cooperation with the bundle that causes the failure. If a failure situation is detected and can be assigned to a certain bundle, then first this bundle should have the chance to take actions to come back to a healthy state. If this is not successful, then appropriate actions must be taken by another entity.

Due to the wide range of potential failures and the definition of resources as very generic and application specific, this can not be achieved by a fixed and inflexible mechanism that handles a fixed set of predefined problems.

What is needed is a flexible framework that allows dynamic provisioning of modules to:

• collect information about resource requirements, and further, the normal, intended states of the monitored entities,
• monitor those resources (as defined above) and ask services for their health status,
• warn interested and legitimate applications when monitored consumptions are above thresholds,
• evaluate the severity of deviations of the currently monitored state from the intended state,
• take decisions and perform actions to recover the intended state,
Less cooperative or legacy applications

In case of an application consumes too much resources, it may affect the quality of service of the other applications installed on the platform. Those situations have to be prevented by OSGi platforms.

As described in the previous chapter, JVMs may provide resource management mechanisms. However, all these solutions are designed to monitor low granularity elements: e.g., threads, classes, objects or methods.

As such, these data are of limited interest and there is a need to raise the abstraction to the primitive deployment unit in OSGi, bundles and applications (or sets of bundles). This encourages the specification of a standard unified OSGi-level API managing resources of bundles and sets of bundles installed on the platform.

This section should be copied from the appropriate RFP(s). It is repeated here so it can be extended while the RFC authors learn more subtle details.

4 Requirements

R1: The solution MUST provide at least one resource accounting algorithm (e.g., direct accounting algorithm).

R2: The solution MUST monitor resources per bundle or per bundle set.

R3: The resource monitoring solution MUST be configurable, enabled and disabled at runtime per bundle or per bundle set.

R4: The solution MUST monitor the following resources, if relevant on the underlying (hardware and software) platform:
   - CPU
   - Memory
   - Disk storage space
   - Bandwidth on any connected network

R5: The solution MUST provide a mechanism to list the resource types that can be monitored on the underlying (hardware and software) platform.

R6: The solution MUST allow the setting of a warning threshold and an error threshold per bundle or set of bundles.

R7: The solution MUST send events while a bundle or a bundle set is exceeding one of the two thresholds defined by R6.

R8: The solution MUST define CPU thresholds as a percentage of use over a configurable period.

R9: The solution MUST define memory thresholds as bytes.

R10: The solution MUST define disk storage space thresholds as bytes.
R11: The solution MUST define thread thresholds as a number of threads.

R12: The solution MUST define socket thresholds as a number of opened sockets.

R13: The solution MUST be able to lower bundle thread priorities while CPU error threshold is reached.

R14: The solution MUST raise an error (e.g., OutOfMemoryError) and MUST prevent further memory allocation while memory error threshold is reached.

R15: The solution MUST raise an error (e.g., IOException) and MUST prevent further disk storage space allocation while disk storage space error threshold is reached.

R16: The solution MUST raise an error (e.g., InternalError) and MUST prevent further thread activation while thread error threshold is reached.

R17: The solution MUST raise an error (e.g., IOException) and MUST prevent further connected-state socket while socket error threshold is reached.

R18: The solution MUST define means for bundles to define their intend resource usage.

R19: The solution MUST allow OSGi applications to monitor bundles, evaluate their states and take decisions to react gracefully.

R20: The solution MAY define optional means for a bundle to resolve its own conflicts based on the decisions of the entity introduced in R19.

R21: Thanks to notification from R6, an application able to monitor the success of R20 MAY take actions, if the conflicts are not resolved after a period of time. Default action MAY be that the framework mechanism resolve this conflict.

R22: The solution MUST provide a mechanism that allows to plug application specific components to evaluate application specific resources.

R23: Special (non standard, see standard Java Runtime definition in section 2) Java platform implementations MAY be necessary to support management of certain resource types.

---

5 Initial Specification Chapter

---

Essentials

Monitoring – Bundle execution resource usage is monitored.

Granular activation – The resource manager can be activated and deactivated per bundle or per bundle set.

Extensibility – five resource types are specified (CPU, memory, disk storage, alive thread and in-use sockets). The list of monitored resource types is extensible and query-able.

Eventing – the resource manager notifies interested entities of exceeded limits.
Entities

Resource Context – A logical entity for resource accounting. A context may be related to a single bundle or a set of bundles.


Platform Resource Context – A Resource context monitoring the resource usage of the platform as a whole.

Resource Monitor – Monitors the usage of a specific resource type for a specific Resource Context. Resource Monitors track resource usage. They hold Resource Thresholds instances. Resource Monitor object implementation may depend on standard or proprietary JVM APIs, and on operating system features.


CPU Monitor – Resource Monitor used to monitor CPU.

Memory Monitor – Resource Monitor used to monitor memory.

Socket Monitor – Resource Monitor used to monitor socket resource.


Thread Monitor – Resource Monitor used to monitor alive Java Thread objects.

Resource Listener – A Resource Listener receives resource threshold notifications.

Resource Event – A Resource Event defines a notification to be sent to Resource Listener instances.

Resource Context Listener – A Resource Context Listener receives notifications about resource context creation and configuration.

Resource Context Event – A Resource Context Event defines a notification to be sent to Resource Context Listeners instances.

Resource Manager – This is a singleton entity which manages Resource Context instances. It is used to create new Resource Context instances and to enumerate existing contexts.

Resource Management Authority – Makes any decision to ensure the quality of the service of the system. They use the Resource Manager to create Resource Context instances. It configures them by adding bundles and Resource Monitors.
Operation summary

Resource Management Authorities use the Resource Manager service to create Resource Contexts. These authorities set bundles or group of bundles to Resource Contexts. They also request every Resource Monitor Factory to create Resource Monitors for a resource type. These Resource Monitors are associated to a single Resource Context.

When activated, Resource Monitors provide the current resource usage per Resource Context. Then, they check whether the current resource usage is compatible with the thresholds held by their associated Resource Listeners. When one of these thresholds is violated, the related Resource Monitor notifies the Resource Listener holding this threshold.

The Resource Manager manages the set of Resource Contexts. Resource Contexts are persistent between platform restarts. Resource Context Listeners are notified when a Resource Context is created or deleted or when a Resource Context configuration (i.e., adding or removing of bundle) is updated.

Figure 2: Resource management class diagram specification
Resource Context

A Resource Context instance is a logical entity used to account resource usage. Every Resource Context defines a bundle scope which can be either a single bundle or a set of bundles. Once the bundle scope is defined, resources used by those bundles are monitored through a set of per-resource-type Resource Monitor instances.

Resource Context instances are persistent. The persistence of those instances is directly managed by the Resource Manager instance.

Each Resource Context is uniquely identified by a name. It can be retrieved through the getName() method. It can not be changed, i.e. it is definitively set when the Resource Context instance is created.

The Resource Context bundle scope is retrieved through the getBundles() method. This bundle scope can be extended through the addBundle(Bundle) method. Bundles can also be removed from a Resource Context through the removeBundle(Bundle, ResourceContext) method. For this last method, a Resource Context instance MAY be specified in order to associate the removed bundle to another Resource Context instance.

Resource Monitor instances are retrieved through getResourceMonitor(String resourceType) method or the getMonitors() method. The list of available resource types is retrieved through the Resource Manager singleton instance.

Resource Monitor instances are added to and removed from a Resource Context instance by calling either addMonitor() method or removeMonitor() method. Both methods SHOULD only be called by ResourceMonitorFactory instances (see ResourceMonitorFactory.createMonitor() method).

A Resource Context is retrieved through the Resource Manager service.

A Resource Context instance can be deleted through removeContext(ResourceContext) method. The Resource Context input argument then defines a destination Resource Context instance for the bundles belonging to the to-be-removed Resource Context instance.

System Resource Context

The System Resource Context is the Resource Context of the core framework. It is retrieved through the Resource Manager service.

The name of this context is “system”.

Framework Resource Context

The Framework Resource Context is a Resource Context monitoring resources of the platform as a whole. It is retrieved through the Resource Manager service. This Resource Context holds all hosted bundles allowing access to the whole platform resource consumption.

The name of this context is “framework”.

Resource Monitor

A Resource Monitor instance monitors a resource type consumed by the bundles of a specific Resource Context instance.

A Resource Context instance holds at most one Resource Monitor instance per monitor-able resource type. Resource Monitor instances are retrieved through their related Resource Context instance. Resource Monitor instances give access to their related Resource Context instance through a call to getContext() method.

The monitored resource type is retrieved through getType() method.

The current usage of a resource consumed by a Resource Context instance is given through getUsage() method. This method returns a Java Object to be casted to the appropriate Java object type depending on the Resource type. The next table provides the expected Java Object type for each specified resource type:
<table>
<thead>
<tr>
<th>Type of Resource</th>
<th>Expected Java Object type</th>
<th>Value description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Long</td>
<td>Cumulative CPU time in ns</td>
</tr>
<tr>
<td>Memory</td>
<td>Long</td>
<td>Allocated memory in bytes</td>
</tr>
<tr>
<td>Threads</td>
<td>Long</td>
<td>Number of alive thread.</td>
</tr>
<tr>
<td>Socket</td>
<td>Long</td>
<td>Number of in-use socket.</td>
</tr>
<tr>
<td>Disk storage space</td>
<td>Long</td>
<td>Bytes on the bundle persistent storage area</td>
</tr>
</tbody>
</table>

For example, for a Memory Monitor instance, a call to MemoryMonitor.getUsage() returns a Long java object indicating the amount of memory the related Resource Context instance is consuming.

A Resource Monitor instance is enabled and disabled through enable() and disable() methods. The state (enabled or disabled) of a Resource Monitor is retrieved through a call to isEnabled() method.

A Resource Monitor instance can also be deleted (delete() method). isDeleted() method returns true if the ResourceMonitor instance has been deleted.

Five types of Resource Monitor are specified:

- CPU Monitor
- Memory Monitor
- Socket Monitor
- Disk Storage Monitor
- Thread Monitor

The support of any Resource Monitor is optional. This list MAY be extended by the solution vendor. The list of the types that are supported on the OSGi platform can be computed by querying ResourceMonitorFactory services.

### Resource Monitor Factory

A Resource Monitor Factory is a service that provides Resource Monitor instances of a specific resource type (e.g., CPUMonitor, MemoryMonitor...) for every Resource Context.

Every Resource Monitor Factory service is registered with the org.osgi.resourcemanager.ResourceType mandatory property. This property indicates which type of Resource Monitor a Resource Monitor Factory is able to create. The type can also be retrieved through a call to ResourceMonitorFactory.getType(). The type MUST be unique (two Resource Monitor Factory instances MUST not have the same type).

New Resource Monitor instances are created by a call to createResourceMonitor(ResourceContext). This method returns a new Resource Monitor instance associated to the provided Resource Context instance (the ResourceMonitorFactory MUST call ResourceContext.addMonitor() to associate the newly created ResourceMonitor with the provided ResourceContext instance). The newly created Resource Monitor is disabled, i.e., it is initially not monitoring the Resource Context resource consumption. It can be activated through a call to ResourceMonitor.enable().

Resource Monitor instances are deleted by calling ResourceMonitor.delete() method.

A Resource Monitor instance MUST only be created through its ResourceMonitorFactory.

Resource Monitor Factory instances should be only used by the Resource Manager singleton instance. The Resource Manager singleton instance performs a service lookup on all existing Resource Monitor Factories. It
uses a Resource Monitor Factory instance when it has to create a new Resource Context instance and their associated Resource Monitor instances.

**CPU Monitor**
A CPU Monitor instance is a Resource Monitor used to monitor the CPU usage of the bundles belonging to a Resource Context.

CPU usage and thresholds are expressed as a cumulative number of nanoseconds (Long). The encapsulated value can be retrieved with the CPUMonitor.getCPUUsage() method.

In case where a threshold is reached, the CPU Monitor instance generates an event triggering Resource Authorities defined corrective actions (e.g., decrease thread priority).

**Memory Monitor**
A Memory Monitor instance monitors and limits the memory used by the bundles of a Resource Context instance.

Memory is accounted as bytes. Memory usage and thresholds are Long java objects. The encapsulated value can be retrieved through the getMemoryUsage() method.

When an error threshold is reached, the next memory allocation MAY throw a MemoryException in the associated context.

**Socket Monitor**
A Socket Monitor instance monitors and limits the number of existing sockets (e.g., TCP, UDP) which are considered to be in use (e.g., listening for incoming packet, bound, or sending outgoing packets).

A Socket is considered to be in-use state when a native socket file descriptor is created. It leaves this state when this socket file descriptor is deleted.

The number of in-use sockets is a Long. The encapsulated value can be retrieved using SocketMonitor.getSocketUsage() method.

When an ERROR threshold is reached, the next socket file descriptor creation in the associated context MAY throw a SocketException.

**Disk Storage Monitor**
A Disk Storage Monitor instance monitors and limits the use of persistent storage within Bundle Persistent Storage Area a Resource Context (the bundles actually belonging to it) consumes.

Disk Storage is expressed as a number of bytes of type Long. The encapsulated value can be retrieved using DiskStorageMonitor.getUsedDiskStorage() method.

A IOException MAY be thrown in the associated context when an error threshold is reached.

**Thread Monitor**
A Thread Monitor instance monitors and limits the number of alive Java Thread objects for a Resource Context instance. A Thread is considered to be alive when it is in the RUNNABLE, BLOCKED, WAITING or TIMED_WAITING java state.

Usage and thresholds are Java Long objects. The encapsulated value can be retrieved using ThreadMonitor.getAliveThreads() method.

When an error threshold is reached, any further thread activation will be prevented in the associated context. An InternalError exception MAY also be thrown in the associated context.
**Resource Listener**

A Resource Listener receives notifications about resource usage for a specific Resource Context and a specific type of resource. A notification will be sent to a Resource Listener when one of its thresholds is violated.

A Resource Listener holds two types of threshold:

- A lower threshold type. This kind of threshold is reached when the monitored resource usage decreases below the threshold.
- An upper threshold type. An upper threshold is reached when the monitored resource usage exceeds this threshold.

Each of them have two levels:

- a WARNING level
- an ERROR level.

A threshold has the following state diagram, which transitions are associated to events:

![Threshold state diagram](image)

**Figure 3: Threshold state diagram**

A threshold state depends on the current consumption of resource and the type of threshold (upper or lower threshold).

A Resource Listener is registered as an OSGi service. The implementer must provide the two following mandatory properties:
• RESOURCE_CONTEXT property – a String defining the name of Resource Context for which the Listener wants to receive threshold notifications.

• RESOURCE_TYPE property – a String defining which type of resource the listener wants to monitor.

It also has to provide at least one of these four properties when registered as an OSGi service:

• UPPER_WARNING_THRESHOLD
• UPPER_ERROR_THRESHOLD
• LOWER_WARNING_THRESHOLD
• LOWER_ERROR_THRESHOLD

These properties are mapped to the four types of threshold values a Resource Listener may support. The service properties are used to notify the associated Resource Monitor when one of these threshold values is modified.

Threshold values can also be retrieved through a set of getter methods. All of these methods return a Comparable object used by the associated Resource Monitor in order to determine the current state of the current usage.

RESOURCE_CONTEXT and RESOURCE_TYPE properties are used by Resource Monitors to identify their associated Resource Listeners. Once associated, a Resource Monitor retrieves the threshold settings using service properties. When one of its thresholds is reached, the Resource Monitor calls ResourceListener.notify(ResourceEvent).

Two examples of resource consumption are explained below, first with in-use sockets monitoring, second with CPU monitoring. The next picture shows the state diagram of the number of in-use state socket over the time.
In our example, the lower warning threshold and the lower error threshold of the Resource Listener are respectively set to 10 and 5. When the number of in-use sockets decreases under 10, the usage goes from the NORMAL state to the WARNING state and the Resource Listener receives a WARNING event. If the number of in-use state sockets decreases again and goes down to 5, the usage goes from the WARNING state to the ERROR state and the Resource Listener receives an ERROR Resource Event.

The upper threshold is also set. The upper warning threshold and the upper error threshold are respectively set to 100 and 1000 in-use state sockets. When the number of sockets reaches 100, the usage goes from the NORMAL state to the WARNING state and the Resource Listener receives a WARNING Resource Event. If this number is still increasing and exceeds 1000, then the usage goes from the WARNING state to the ERROR state and the Resource Listener receives an ERROR Resource Event.

This is a typical use case for a Java Web server. Indeed, one of the most important quality of service indicator is the number of in-use state sockets a Java web server is handling. A low number of in-use state sockets may indicate the Java web server encounters network problems. On the contrary, a high number of in-use state socket may be the result of an external network attack or it could also indicates the Java web server is overused and its administrator should take actions to load-balance the charge to another Java web server instance.

For other resource types, only upper thresholds may be useful. The next diagram shows the CPU consumption a Resource Context is using over the time:

---

![CPU consumption (%) over the time – Upper Threshold](image-url)

- **State:**
  - **ERROR**
  - **WARNING**
  - **NORMAL**

- **Thresholds:**
  - **Error threshold**
  - **Warning threshold**

- **Events:**
  - **Events are emitted**

---

An Authority takes action in order to preserve the QOS: here it stops the bundle.
In this example, only the upper threshold is set. The upper warning threshold is set to 50%, the error one is set to 75%. CPU consumption fluctuates between 0 and 50%, the usage is in the NORMAL state. Then it increases and reaches 50%. The usage then goes from the NORMAL state to the WARNING state and the Resource Listener holding the threshold receives a WARNING Resource Event.

Afterwards, CPU consumption decreases under 50%; the usage goes from the WARNING state to the NORMAL state. The related Resource listener receives a NORMAL Resource Event.

It then increases again and exceeds 50%. The usage goes to the WARNING state. CPU consumption is still increasing and exceeds 75%. At this moment, the usage goes from the WARNING state to the ERROR state and the related Resource Listener receives an ERROR Resource Event.

After some seconds in the ERROR state, the Resource Listener implementation stops the bundle in order to preserve the quality of service.

The choice of the type of threshold (lower or upper, or both of them) depends on the type of resource and the needs of the Resource Management Authorities providing the Resource Listener. Other resources like the free memory may take advantage of a lower threshold.

**Resource Event**

A Resource Event instance is an event sent to a Resource Listener when one of its thresholds is reached. This event is notified to a Resource Listener through a call to ResourceListener.notify(ResourceEvent).

A Resource Event has a type among the following ones:

- **ERROR** – The resource consumption reaches either the upper or the lower error threshold of the Resource Listener receiving this event,
- **WARNING** – The resource consumption reaches either the upper or the lower warning threshold of the Resource Listener receiving this event.
- **NORMAL** – The resource consumption is back from warning or error state to normal state.

The Resource Listener instance analyzes this event by calling the following methods:

- `getValue()` method returns the resource consumption at the time when the Resource Event instance was generated.
- `isUpperThreshold()` method returns true if the reached threshold is an upper threshold type. If this method returns false, this is a lower threshold.
- `getType()` method indicates the state (WARNING, ERROR, or NORMAL) of the resource usage.
- `getContext()` method returns the Resource Context instance related to this event. The Resource Listener can use it to retrieve the Resource Monitor instance (e.g., event.getContext().getMonitor(event.getResourceType())).

**Resource Context Listener**

A Resource Listener instance receives notifications about Resource Context lifecycle and configuration.

A notification will be sent when:

- A Resource Context is created.
- A Resource Context is updated, i.e., a bundle has been added or removed from a Resource Context instance.
- A Resource Context is deleted.
An application which is interested in notifications has to register a Resource Context Listener instance as an OSGi service. The application may provide a set of properties at registration time to reduce the number of notifications a Resource Listener instance will receive. The available properties are:

- **RESOURCE_CONTEXT** property – An array of String defining the name of Resource Context instances. If defined, a Resource Listener instance will only receive notifications related to these specified Resource Context instances.
- **RESOURCE_TYPE** property – an array of integers defining the type of notifications a Resource Context Listener instance will receive (see types defined in section Resource Context Event).

A Resource Context Listener instance is notified through a call to notify(ResourceContextEvent) method.

## Resource Context Event

A Resource Context Event instance is an event sent to Resource Context Listener instances through a call to ResourceContextListener.notify(ResourceContextEvent) method.

A Resource Context Event has a type among the four following ones:

- **RESOURCE_CONTEXT_CREATED** – A new Resource Context instance has been created.
- **RESOURCE_CONTEXT_REMOVED** – A Resource Context instance has been deleted.
- **BUNDLE_ADDED** – A bundle has been added in the scope of a Resource Context instance
- **BUNDLE_REMOVED** – A bundle has been removed from the scope of a Resource Context instance.

In the case of a RESOURCE_CONTEXT_ADDED event or a RESOURCE_CONTEXT_REMOVED event, a call to getContext() returns the targeted Resource Context instance.

In the case of a BUNDLE_ADDED type or BUNDLE_REMOVED type, getBundle() returns the Bundle object to be added to or removed from. The related Resource Context instance is given by a call to getContext().

## Resource Manager

The Resource Manager service manages the Resource Context instances. The Resource Manager service is available through the OSGi service registry.

This service holds the existing Resource Context instances. Resource Context instances are created by calling the createContext(String, ResourceContext) method. The caller provides a context name as a string and optionally a template as a ResourceContext object.

The list of existing Resource Context instances can be retrieved through the following methods:

- **getContext(String)** – retrieve a Resource Context instance by name.
- **getContext(Thread)** – retrieve the Resource Context instance related to a Thread.
- **getCurrentContext()** – retrieve the Resource Context instance based on the current thread.
- **listContexts()** - retrieve all existing Resource Context instances as an array.

The Resource Manager singleton manages the persistence of the Resource Context instances. The following properties are stored:

- name of the Resource Context.
- list of the bundles belonging to the Resource Context.
- list of the Resource Monitor instances. For each one:
  - sampling period.
The way the Resource Manager persists the Resource Context instances is implementation specific. The implementer is free to use any file format and file location it wants. At startup, the Resource Manager will load the persisted Resource Context instances to restore shutdown state.

### Resource Management Authority

A Resource Management Authority uses the Resource Manager singleton instance to apply Resource Management policies. These entities MAY:

- create and configure Resource Context instances (resource thresholds, bundle scope)
- take any decisions (stop a bundle, uninstall a bundle) if a Resource Context exceeds resource limit.

These policies are out of the scope of this specification.

### 6 Javadoc
<table>
<thead>
<tr>
<th>Package Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.service.resourcemangement</td>
<td>28</td>
</tr>
<tr>
<td>org.osgi.service.resourcemangement.monitor</td>
<td>59</td>
</tr>
</tbody>
</table>
## Interface Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceContext</td>
<td>Logical entity for resource accounting.</td>
<td>30</td>
</tr>
<tr>
<td>ResourceContextListener</td>
<td>A <code>ResourceContextListener</code> is notified whenever:</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>- a <code>ResourceContext</code> is created or deleted.</td>
<td></td>
</tr>
<tr>
<td>ResourceListener</td>
<td>A <code>ResourceListener</code> is an OSGi service which is notified when a <code>ResourceContext</code> violates one of the threshold defined by the listener.</td>
<td>43</td>
</tr>
<tr>
<td>ResourceManager</td>
<td>Monitor and manage resource contexts.</td>
<td>47</td>
</tr>
<tr>
<td>ResourceMonitor</td>
<td>Representation of the state of a resource for a <code>ResourceContext</code>.</td>
<td>51</td>
</tr>
<tr>
<td>ResourceMonitorFactory</td>
<td>A <code>ResourceMonitorFactory</code> are OSGi services used to create <code>ResourceMonitor</code> instance.</td>
<td>56</td>
</tr>
</tbody>
</table>

## Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceContextEvent</td>
<td>An event sent to <code>ResourceListener</code> when resource usage violates one of their thresholds.</td>
<td>34</td>
</tr>
<tr>
<td>ResourceEvent</td>
<td>An event sent to <code>ResourceListener</code> when resource usage violates one of their thresholds.</td>
<td>40</td>
</tr>
</tbody>
</table>

## Exception Summary

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemoryException</td>
<td>This exception is used to report that a <code>ResourceContext</code> has reached an <code>ERROR</code> threshold about memory.</td>
<td>29</td>
</tr>
<tr>
<td>ResourceMonitorException</td>
<td>Resource Monitor Exception reports an invalid usage of a <code>ResourceContext</code>.</td>
<td>55</td>
</tr>
<tr>
<td>ThresholdException</td>
<td>Threshold exception are thrown an invalid threshold setting is detected.</td>
<td>58</td>
</tr>
</tbody>
</table>
Class MemoryException

org.osgi.service.resourcemanagement

java.lang.Object
   java.lang.Throwable
      java.lang.Exception
         org.osgi.service.resourcemanagement.MemoryException

All Implemented Interfaces:
   Serializable

public class MemoryException
   extends RuntimeException

This exception is used to report that a Resource Context has reached an ERROR threshold about memory.

Immutable

Constructor Summary

MemoryException(long pMemoryUsage)

Create a new MemoryException

Parameters:
   pMemoryUsage - the memory consumption at the moment when this exception was thrown

Method Summary

long getMemoryUsage()

Retrieves the memory consumption at the moment when this exception was thrown.

Constructor Detail

MemoryException

public MemoryException(long pMemoryUsage)

Create a new MemoryException

Parameters:
   pMemoryUsage - the memory consumption at the moment when this exception was thrown

Method Detail

getMemoryUsage

public long getMemoryUsage()

Retrieves the memory consumption at the moment when this exception was thrown.

Returns:
   memory consumption.
Interface ResourceContext

org.osgi.service.resourcemangement

public interface ResourceContext

Logical entity for resource accounting. A resource context has a group of member bundles, and a bundle can be a member of 1 or 0 resource contexts.

Management agents can use the `ResourceManager createContext(String, ResourceContext)` method to create ResourceContext instances.

Management agents can use the `getMonitor(String)` method to get `ResourceMonitor` instances for the supported resource types. These instances can then be used to monitor the usage of the resources, or the set usage limits.

ResourceContexts are retrieved through the `ResourceManager` OSGi service.

ThreadSafe

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void addBundle(long bundleId)</code></td>
<td>Adds a bundle to the resource context.</td>
<td>31</td>
</tr>
<tr>
<td><code>void addResourceMonitor(ResourceMonitor resourceMonitor)</code></td>
<td>Add a new ResourceMonitor instance monitoring resource for this resource context.</td>
<td>32</td>
</tr>
<tr>
<td><code>boolean equals(Object resourceContext)</code></td>
<td>A ResourceContext rc1 is equals to ResourceContext rc2 if rc1.getName() is equals to rc2.getName().</td>
<td>33</td>
</tr>
<tr>
<td><code>long[] getBundleIds()</code></td>
<td>Returns the bundle identifiers belonging to this Resource Context.</td>
<td>31</td>
</tr>
<tr>
<td><code>ResourceMonitor getMonitor(String resourceType)</code></td>
<td>Returns a ResourceMonitor instance for the specified resource type.</td>
<td>32</td>
</tr>
<tr>
<td><code>ResourceMonitor[] getMonitors()</code></td>
<td>Retrieves all the existing ResourceMonitor belonging to this context.</td>
<td>32</td>
</tr>
<tr>
<td><code>String getName()</code></td>
<td>Returns the name of the resource context.</td>
<td>31</td>
</tr>
<tr>
<td><code>int hashCode()</code></td>
<td>Retrieves the hashCode value of a ResourceContext.</td>
<td>33</td>
</tr>
<tr>
<td><code>void removeBundle(long bundleId)</code></td>
<td>Removes the bundle identified by bundleId from the Resource Context.</td>
<td>31</td>
</tr>
<tr>
<td><code>void removeBundle(long bundleId, ResourceContext destination)</code></td>
<td>Removes the bundle from this resource context.</td>
<td>31</td>
</tr>
<tr>
<td><code>void removeContext(ResourceContext destination)</code></td>
<td>Removes a resource context.</td>
<td>33</td>
</tr>
<tr>
<td><code>void removeResourceMonitor(ResourceMonitor resourceMonitor)</code></td>
<td>Remove a ResourceMonitor instance from the context.</td>
<td>32</td>
</tr>
</tbody>
</table>
Method Detail

getName

String getName()

Returns the name of the resource context. Resource context names are unique within a framework instance.

Returns: The resource context name

getBundleIds

long[] getBundleIds()

Returns the bundle identifiers belonging to this Resource Context.

Returns: An array of org.osgi.framework.Bundle objects, or an empty array if no bundles are currently members of this context

addBundle

void addBundle(long bundleId)

Adds a bundle to the resource context. The bundle will be a member of the context until it is uninstalled, or explicitly removed from the context with removeBundle(long) method or removeBundle(long, ResourceContext) method.

Resources previously allocated by this bundle (in another resource context) will not be moved to this resource context. The change applies only for future allocations.

A ResourceContextEvent with type ResourceContextEvent.BUNDLE_ADDED will be sent.

Parameters:
  bundleId - The bundle to add to this resource context

removeBundle

void removeBundle(long bundleId)

Removes the bundle identified by bundleId from the Resource Context. The bundle is no longer to this Resource Context.

Parameters:
  bundleId - bundle identifier

removeBundle

void removeBundle(long bundleId, ResourceContext destination)
Interface ResourceContext

Removes the bundle from this resource context. If a destination context is specified, the bundle will be added in it.

Resources previously allocated by this bundle will not be removed from the resource context. The change applies only for future allocations.

A ResourceContextEvent with type ResourceContextEvent.BUNDLE_REMOVED will be sent.

Parameters:
- bundleId - the identifier of the bundle to be removed from the Resource Context
- destination - A resource context in which to add the bundle, after removing it from this context. If no destination is provided (i.e. null), the bundle is not associated to a new Resource Context.

getMonitor

ResourceMonitor getMonitor(String resourceType)

Returns a ResourceMonitor instance for the specified resource type. If the ResourceManager implementation does not support this resource type, null is returned.

Parameters:
- resourceType - The resource type, for which a resource monitor is requested

Returns:
- A ResourceMonitor instance, or null, if this resource type is not supported

getMonitors

ResourceMonitor[] getMonitors()

Retrieves all the existing ResourceMonitor belonging to this context.

Returns:
- an array of ResourceMonitor. May be empty if no ResourceMonitor

addResourceMonitor

void addResourceMonitor(ResourceMonitor resourceMonitor)

Add a new ResourceMonitor instance monitoring resource for this resource context. This method should be called only by ResourceMonitorFactory instance.

Parameters:
- resourceMonitor - resourceMonitor instance to be added

Throws:
- ResourceMonitorException - if resourceMonitor is associated to another context or resourceMonitor has been deleted.

removeResourceMonitor

void removeResourceMonitor(ResourceMonitor resourceMonitor)

Remove a ResourceMonitor instance from the context.
Interface ResourceContext

Parameters:
resourceMonitor - resource monitor instance to be removed

removeContext

void removeContext(ResourceContext destination)

Removes a resource context. All resources allocated in this resource context will be moved to the destination context. If destination is null, these resources will no longer be monitored.

A ResourceContextEvent with type ResourceContextEvent.RESOURCE_CONTEXT_DELETED will be sent.

Parameters:
destination - The ResourceContext where the resources currently allocated by this resource context will be moved.

equals

boolean equals(Object resourceContext)

A ResourceContext rc1 is equals to ResourceContext rc2 if rc1.getName() is equals to rc2.getName().

Overrides: equals in class Object

Parameters:
resourceContext - resource context

Returns:
true if getName().equals(resourceContext.getName())

hashCode

int hashCode()

Retrieves the hashCode value of a ResourceContext. The hashCode value of a ResourceContext is only based on the hashcode value of the name of the context.

Overrides: hashCode in class Object

Returns:
hashcode
**Class ResourceContextEvent**

gov.osgi.service.resourcemanager

java.lang.Object

org.osgi.service.resourcemanager.ResourceContextEvent

public class ResourceContextEvent
extends Object

Immutable

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int</td>
<td>BUNDLE_ADDED</td>
</tr>
<tr>
<td>static int</td>
<td>BUNDLE_REMOVED</td>
</tr>
<tr>
<td>static int</td>
<td>RESOURCE_CONTEXT_CREATED</td>
</tr>
<tr>
<td>static int</td>
<td>RESOURCE_CONTEXT_DELETED</td>
</tr>
</tbody>
</table>

### Constructor Summary

- **ResourceContextEvent** (int pType, ResourceContext pResourceContext)
  Create a new ResourceContextEvent.

- **ResourceContextEvent** (int pType, ResourceContext pResourceContext, long pBundleId)
  Create a new ResourceContextEvent.

### Method Summary

- boolean equals (Object var0)
- long getBundleId()
  Retrieves the identifier of the bundle being added to or removed from the Resource Context.
- ResourceContext getContext()
  Retrieves the Resource Context associated to this event
- int getType()
  Retrieves the type of this Resource Context Event.
- int hashCode()
- String toString()
### Field Detail

**RESOURCE_CONTEXT_CREATED**

```java
public static final int RESOURCE_CONTEXT_CREATED = 0
```

A new `ResourceContext` has been created.

The `ResourceManager.createContext(String, ResourceContext)` method has been invoked.

**RESOURCE_CONTEXT_DELETED**

```java
public static final int RESOURCE_CONTEXT_DELETED = 1
```

A `ResourceContext` has been deleted.

The `ResourceContext.removeContext(ResourceContext)` method has been invoked.

**BUNDLE_ADDED**

```java
public static final int BUNDLE_ADDED = 2
```

A bundle has been added to a `ResourceContext`.

The `ResourceContext.addBundle(long)` method has been invoked.

**BUNDLE_REMOVED**

```java
public static final int BUNDLE_REMOVED = 3
```

A bundle has been removed from a `ResourceContext`.

The `ResourceContext.removeBundle(long)` method or `ResourceContext.removeBundle(long, ResourceContext)` method have been invoked, or the bundle has been uninstalled.

### Constructor Detail

**ResourceContextEvent**

```java
public ResourceContextEvent(int pType, ResourceContext pResourceContext)
```

Create a new `ResourceContextEvent`. This constructor should be used when the type of the event is either `RESOURCE_CONTEXT_CREATED` or `{1}.

**Parameters:**

- `pType` - event type
- `pResourceContext` - context
Create a new ResourceContextEvent. This constructor should be used when the type of the event is either 
`BUNDLE_ADDED` or `BUNDLE_REMOVED`.

**Parameters:**
- `pType` - event type
- `pResourceContext` - context
- `pBundleId` - bundle

**Method Detail**

**getType**

```java
public int getType()
```

Retrieves the type of this Resource Context Event.

**Returns:**
- the type of the event. One of:
  - `RESOURCE_CONTEXT_CREATED`
  - `RESOURCE_CONTEXT_DELETED`
  - `BUNDLE_ADDED`
  - `BUNDLE_REMOVED`

**getContext**

```java
public ResourceContext getContext()
```

Retrieves the Resource Context associated to this event

**Returns:**
- Resource Context.

**getBundleId**

```java
public long getBundleId()
```

Retrieves the identifier of the bundle being added to or removed from the Resource Context.

This method returns a valid value only when `getType()` returns:

- `BUNDLE_ADDED`
- `BUNDLE_REMOVED`

**Returns:**
- the bundle id or -1 (invalid value).
**Class ResourceContextEvent**

**toString**

public String toString()

*Overrides:*

`toString` in class `Object`

**hashCode**

public int hashCode()

*Overrides:*

`hashCode` in class `Object`

**equals**

public boolean equals(Object var0)

*Overrides:*

`equals` in class `Object`
public interface ResourceContextListener

A ResourceContextListener is notified whenever:

- a ResourceContext is created or deleted.
- a bundle is added or removed from a ResourceContext.

A ResourceContextListener is registered as an OSGi service. At the registration time, the following properties may be provided:

- the RESOURCE_CONTEXT property which limits the Resource Context for which notifications will be received. This property can be either a String value or an array of String. If this property is not set, the Resource Context Listener receives events from all the Resource Context.
- the EVENT_TYPE property. If set, this property filters the type of event this listener will receive. The value of this property can either a String (then, the Listener receives notifications about a single type) or an array of String (several types). The expected values for this property are the types defined by a ResourceContextEvent:
  - ResourceContextEvent.RESOURCE_CONTEXT_CREATED
  - ResourceContextEvent.RESOURCE_CONTEXT_DELETED
  - ResourceContextEvent.BUNDLE_ADDED
  - ResourceContextEvent.BUNDLE_REMOVED

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_TYPE</td>
<td>String</td>
<td>Property specifying the type of events this listener will receive.</td>
<td>38</td>
</tr>
<tr>
<td>RESOURCE_CONTEXT</td>
<td>String</td>
<td>Property specifying the ResourceContext(s) for which a notification will be received by this listener.</td>
<td>38</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>void notify</td>
<td>Notify this listener about a ResourceContext events.</td>
<td>39</td>
</tr>
</tbody>
</table>

### Field Detail

**RESOURCE_CONTEXT**

public static final String RESOURCE_CONTEXT = "resource.context"

Property specifying the ResourceContext(s) for which a notification will be received by this listener.

The property value is either a string (i.e the name of the ResourceContext) and an array of string (several ResourceContext).

**EVENT_TYPE**

public static final String EVENT_TYPE = "event.type"
Property specifying the type of events this listener will receive. The expected values are the ResourceContextEvent types. It can be a String or an array of String.

**Method Detail**

**notify**

void notify(ResourceContextEvent event)

Notify this listener about a ResourceContext events.

**Parameters:**

- event - event.
public class ResourceEvent
extends Object

An event sent to ResourceListener when resource usage violates one of their thresholds.

ResourceEvent objects are delivered synchronously to all matching ResourceListener services. A type code is used to identify the event.

See Also: ResourceListener
Immutable

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int ERROR</td>
<td></td>
</tr>
<tr>
<td>Type of ResourceEvent indicating a threshold goes to the ERROR state.</td>
<td>41</td>
</tr>
<tr>
<td>static int NORMAL</td>
<td></td>
</tr>
<tr>
<td>Type of ResourceEvent indicating a threshold goes to the NORMAL state.</td>
<td>41</td>
</tr>
<tr>
<td>static int WARNING</td>
<td></td>
</tr>
<tr>
<td>Type of ResourceEvent indicating a threshold goes to the WARNING state.</td>
<td>41</td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceEvent(int pType, ResourceContext pContext, boolean pIsUpperThreshold, Object pValue)</td>
<td>41</td>
</tr>
<tr>
<td>Create a new ResourceEvent.</td>
<td></td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean equals(Object var0)</td>
<td>42</td>
</tr>
<tr>
<td>ResourceContext getContext()</td>
<td></td>
</tr>
<tr>
<td>Returns the resource context that caused the event.</td>
<td>42</td>
</tr>
<tr>
<td>int getType()</td>
<td></td>
</tr>
<tr>
<td>Returns the event type.</td>
<td>41</td>
</tr>
<tr>
<td>Object getValue()</td>
<td></td>
</tr>
<tr>
<td>Returns the resource consumption value.</td>
<td>41</td>
</tr>
<tr>
<td>int hashCode()</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>boolean isUpperThreshold()</td>
<td></td>
</tr>
<tr>
<td>Returns true if the threshold triggering this event is an upper threshold.</td>
<td>42</td>
</tr>
<tr>
<td>String toString()</td>
<td></td>
</tr>
</tbody>
</table>

OSGi Javadoc -- 30/07/14
Class ResourceEvent

Field Detail

NORMAL

public static final int NORMAL = 0

Type of ResourceEvent indicating a threshold goes to the NORMAL state.

WARNING

public static final int WARNING = 1

Type of ResourceEvent indicating a threshold goes to the WARNING state.

ERROR

public static final int ERROR = 2

Type of ResourceEvent indicating a threshold goes to the ERROR state.

Constructor Detail

ResourceEvent

public ResourceEvent(int pType,
        ResourceContext pContext,
        boolean pIsUpperThreshold,
        Object pValue)

Create a new ResourceEvent.

Method Detail

getType

public int getType()

Returns the event type. The type values are:

- NORMAL
- WARNING
- ERROR

Returns:
The event type

getValue

public Object getValue()

Returns the resource consumption value. Relevant only for event types NORMAL, WARNING and ERROR.
Class ResourceEvent

Returns:
the resource consumption value, or null if a resource monitor is not relevant

getContext

public ResourceContext getContext()

Returns the resource context that caused the event.

Returns:
The resource context that cased the event.

isUpperThreshold

public boolean isUpperThreshold()

Returns true if the threshold triggering this event is an upper threshold. This method is only used when getType() returns NORMAL, WARNING or ERROR.

Returns:
true if it is an upper threshold.

hashCode

public int hashCode()

Overrides:
hashCode in class Object

equals

public boolean equals(Object var0)

Overrides:
equals in class Object

toString

public String toString()

Overrides:
toString in class Object
public interface ResourceListener

A ResourceListener is an OSGi service which is notified when a Resource Context violates one of the threshold defined by the listener.

Every ResourceListener is associated to a specific Resource Context and a specific Resource type. It defines two types of thresholds: a lower and an upper. A lower threshold is reached when the resource usage decreases below the threshold. On the contrary, an upper threshold is reached when the resource usage exceeds the threshold.

Both lower or upper threshold are two levels: a warning level and an error level. The warning level indicates the resource usage becomes to be critical but are still acceptable. The error level indicates the resource usage is now critical for the overall system and actions should be taken.

A Resource Listener is registered with these two mandatory properties:

- RESOURCE_CONTEXT which defines the ResourceContext associated to this Listener
- RESOURCE_TYPE which the type of resource

The next optional properties are used to specify threshold values. A ResourceListener must at least provides one of them:

- UPPER WARNING_THRESHOLD
- UPPER ERROR_THRESHOLD
- LOWER WARNING_THRESHOLD
- LOWER ERROR_THRESHOLD

These threshold values can also be retrieved through methods.

Resource Listeners are associated to a Resource Context and a Resource Monitor based on the RESOURCE_CONTEXT property and the RESOURCE_TYPE property (both of them are mandatory at registration time).

Once associated, the ResourceMonitor gets the threshold values through the service properties (i.e UPPER WARNING_THRESHOLD, UPPER ERROR_THRESHOLD, LOWER WARNING_THRESHOLD and LOWER ERROR_THRESHOLD) and store them. Once it detects a new resource consumption, it compares the new resource usage value with the thresholds provided by the Resource Listener. If the resource usage violates one of these thresholds, the Resource Monitor notifies the ResourceListener through a call to notify(ResourceEvent).

A ResourceMonitor tracks threshold value modification by using a org.osgi.framework.ServiceListener.

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String LOWER_ERROR_THRESHOLD</td>
<td>45</td>
</tr>
<tr>
<td>Optional property defining the value of the lower error threshold.</td>
<td></td>
</tr>
<tr>
<td>String LOWER_WARNING_THRESHOLD</td>
<td>44</td>
</tr>
<tr>
<td>Optional property defining the value of the lower warning threshold.</td>
<td></td>
</tr>
<tr>
<td>String RESOURCE_CONTEXT</td>
<td>44</td>
</tr>
<tr>
<td>Mandatory property specifying the Resource Context associated with the listener.</td>
<td></td>
</tr>
<tr>
<td>String RESOURCE_TYPE</td>
<td>44</td>
</tr>
<tr>
<td>Mandatory property defining the type of Resource (i.e the ResourceMonitor) associated to this Listener.</td>
<td></td>
</tr>
<tr>
<td>String UPPER_ERROR_THRESHOLD</td>
<td>44</td>
</tr>
<tr>
<td>Optional property defining the value of the upper error threshold.</td>
<td></td>
</tr>
</tbody>
</table>
String **UPPER_WARNING_THRESHOLD**
Optional property defining the value of the upper warning threshold.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| Comparable | **getLowerErrorThreshold()**
Retrieves the lower error threshold value set by the listener. |
| Comparable | **getLowerWarningThreshold()**
Retrieves the lower warning threshold value set by the listener. |
| Comparable | **getUpperErrorThreshold()**
Retrieves the upper error threshold value set by this listener. |
| Comparable | **getUpperWarningThreshold()**
Retrieves the upper warning threshold value set by this listener. |
| void | **notify(ResourceEvent event)**
Receives a resource management notification |

Field Detail

**RESOURCE_CONTEXT**

```
public static final String RESOURCE_CONTEXT = "resource.context"
```

Mandatory property specifying the Resource Context associated with the listener.

**RESOURCE_TYPE**

```
public static final String RESOURCE_TYPE = "resource.type"
```

Mandatory property defining the type of Resource (i.e the ResourceMonitor) associated to this Listener.

**UPPER_WARNING_THRESHOLD**

```
public static final String UPPER_WARNING_THRESHOLD = "upper.warning.threshold"
```

Optional property defining the value of the upper warning threshold.

**UPPER_ERROR_THRESHOLD**

```
public static final String UPPER_ERROR_THRESHOLD = "upper.error.threshold"
```

Optional property defining the value of the upper error threshold.

**LOWER.WARNING_THRESHOLD**

```
public static final String LOWER_WARNING_THRESHOLD = "lower.warning.threshold"
```

Optional property defining the value of the lower warning threshold.
**LOWER_ERROR_THRESHOLD**

public static final String LOWER_ERROR_THRESHOLD = "lower.error.threshold"

Optional property defining the value of the lower error threshold.

### Method Detail

**notify**

```java
void notify(ResourceEvent event)
```

Receives a resource management notification

**Parameters:**
- `event` - The `ResourceEvent` object

---

**getLowerWarningThreshold**

```java
Comparable getLowerWarningThreshold()
```

Retrieves the lower warning threshold value set by the listener. If the resource usage decreases under this threshold value, the `notify(ResourceEvent)` will be called. The provided `ResourceEvent` then indicates the WARNING state is reached.

**Returns:**
- a comparable object or null if no threshold is set.

---

**getLowerErrorThreshold**

```java
Comparable getLowerErrorThreshold()
```

Retrieves the lower error threshold value set by the listener. If the resource usage decreases under this threshold, the `notify(ResourceEvent)` will be called. The provided `ResourceEvent` then indicates the ERROR state is reached.

**Returns:**
- a comparable object or null if no threshold is set.

---

**getUpperWarningThreshold**

```java
Comparable getUpperWarningThreshold()
```

Retrieves the upper warning threshold value set by this listener. If the resource usage exceeds this threshold, the `notify(ResourceEvent)` method will be called. The provided `ResourceEvent` then indicates the WARNING state is reached.

**Returns:**
- a comparable object or null if no threshold is reached.
getUpperErrorThreshold

Comparable `getUpperErrorThreshold()`

Retrieves the upper error threshold value set by this listener. If the resource usage exceeds this threshold, the `notify(ResourceEvent)` will be called. The provided `ResourceEvent` then indicates the ERROR state is reached.

**Returns:**

- a comparable object or null if no threshold is reached.
public interface ResourceManager

Monitor and manage resource contexts.

The ResourceManager is a service OSGi.

ThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>String</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAMEWORK_CONTEXT_NAME</td>
<td>48</td>
<td>The name of the special, optional resource context, representing the whole OSGi framework.</td>
</tr>
<tr>
<td>RES_TYPE_CPU</td>
<td>48</td>
<td>The name of the CPU resource type, used to monitor and control the CPU time used by a resource context.</td>
</tr>
<tr>
<td>RES_TYPE_DISK_STORAGE</td>
<td>48</td>
<td>The name of the disk storage resource type, used to monitor and control the size of the persistent storage used by a resource context.</td>
</tr>
<tr>
<td>RES_TYPE_MEMORY</td>
<td>48</td>
<td>The name of the memory resource type, used to monitor and control the size of the java heap used by a resource context.</td>
</tr>
<tr>
<td>RES_TYPE_SOCKET</td>
<td>48</td>
<td>The name of the socket resource type, used to monitor and control the number of existing sockets used by a resource context.</td>
</tr>
<tr>
<td>RES_TYPE_THREADS</td>
<td>48</td>
<td>The name of the threads resource type, used to monitor and control the number of threads created by a resource context.</td>
</tr>
<tr>
<td>SYSTEM_CONTEXT_NAME</td>
<td>48</td>
<td>The name of the Resource Context associated with System bundle (bundle 0).</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>ResourceManager</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createContext(String name, ResourceContext template)</td>
<td>49</td>
<td>Creates a new ResourceContext.</td>
</tr>
<tr>
<td>getContext(String name)</td>
<td>49</td>
<td>Returns the context with the specified name.</td>
</tr>
<tr>
<td>getContext(long bundleId)</td>
<td>49</td>
<td>Returns the ResourceContext associated to the provided bundle.</td>
</tr>
<tr>
<td>getSupportedTypes()</td>
<td>50</td>
<td>Returns a list with the supported resource type names.</td>
</tr>
<tr>
<td>listContext()</td>
<td>49</td>
<td>Lists all available resource contexts.</td>
</tr>
</tbody>
</table>
Field Detail

RES_TYPE_THREADS

public static final String RES_TYPE_THREADS = "resource.type.threads"

The name of the threads resource type, used to monitor and control the number of threads created by a resource context. ResourceManager implementations must create ThreadMonitor instances for this resource type.

RES_TYPE_CPU

public static final String RES_TYPE_CPU = "resource.type.cpu"

The name of the CPU resource type, used to monitor and control the CPU time used by a resource context. ResourceManager implementations must create CPUMonitor instances for this resource type.

RES_TYPE_DISK_STORAGE

public static final String RES_TYPE_DISK_STORAGE = "resource.type.disk.storage"

The name of the disk storage resource type, used to monitor and control the size of the persistent storage used by a resource context. ResourceManager implementations must create DiskStorageMonitor instances for this resource type.

RES_TYPE_MEMORY

public static final String RES_TYPE_MEMORY = "resource.type.memory"

The name of the memory resource type, used to monitor and control the size of the java heap used by a resource context. ResourceManager implementations must create MemoryMonitor instances for this resource type.

RES_TYPE_SOCKET

public static final String RES_TYPE_SOCKET = "resource.type.socket"

The name of the socket resource type, used to monitor and control the number of existing sockets used by a resource context. ResourceManager implementations must create SocketMonitor instances for this resource type.

FRAMEWORK_CONTEXT_NAME

public static final String FRAMEWORK_CONTEXT_NAME = "framework"

The name of the special, optional resource context, representing the whole OSGi framework.

SYSTEM_CONTEXT_NAME

public static final String SYSTEM_CONTEXT_NAME = "system"
Interface ResourceManager

The name of the Resource Context associated with System bundle (bundle 0).

**Method Detail**

**listContext**

```java
ResourceContext[] listContext()
```

Lists all available resource contexts. The list will contain the special FRAMEWORK_CONTEXT_NAME context and the SYSTEM_CONTEXT_NAME context, if it is supported.

**Returns:**
An array of ResourceContext objects, or an empty array, if no contexts have been created.

**createContext**

```java
ResourceContext createContext(String name,
                               ResourceContext template)
```

Creates a new ResourceContext.

A ResourceContextEvent with type ResourceContextEvent.RESOURCE_CONTEXT_CREATED will be sent.

**Parameters:**
- **name**: The name identifying the context. Names must be unique within the framework instance.
- **template**: If a template is provided, the new resource context will inherit all resource monitoring settings (enabled monitors, thresholds) from the template.

**Returns:**
A new ResourceContext instance.

**getContext**

```java
ResourceContext getContext(String name)
```

Returns the context with the specified name

**Parameters:**
- **name**: The resource context name

**Returns:**
An existing ResourceContext with the specified name, or null if such a context doesn't exist

**getContext**

```java
ResourceContext getContext(long bundleId)
```

Returns the ResourceContext associated to the provided bundle

**Parameters:**
- **bundleId**: bundle identifier

**Returns:**
the ResourceContext associated to bundle b or null if the bundle b does not belong to a Resource Context.
Interface ResourceManager

getSupportedTypes

String[] getSupportedTypes()

Returns a list with the supported resource type names.

Returns:
An array containing the names of all resource types that this ResourceManager implementation supports.
Interface ResourceMonitor

org.osgi.service.resourcemangement

All Known Subinterfaces:
CPUMonitor, DiskStorageMonitor, MemoryMonitor, SocketMonitor, ThreadMonitor

public interface ResourceMonitor

Representation of the state of a resource for a resource context.

ResourceMonitor objects are returned by the ResourceContext.getMonitor(String) method.

The ResourceMonitor object may be used to:

- Enable/Disable the monitoring of the corresponding resource type for the corresponding resource context
- View the current usage of the resource by this resource context

A resource monitor can have a sampling period, a monitored period, or both. For example, for CPU monitoring, the resource management implementation can get the CPU usage of the running threads once per minute, and calculate the CPU usage per context in percentages based on the last ten such measurements. This will make a 60 000 milliseconds sampling period, and a 600 000 milliseconds monitored period.

ThreadSafe

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>void delete() Disable and delete this instance of Resource Monitor.</td>
<td>52</td>
</tr>
<tr>
<td>void disable() Disable the monitoring of this resource type for the resource context associated with this monitor instance.</td>
<td>53</td>
</tr>
<tr>
<td>void enable() Enable the monitoring of this resource type for the resource context associated with this monitor instance.</td>
<td>53</td>
</tr>
<tr>
<td>boolean equals(Object resourceMonitor) Checks if resourceMonitor is equals to the current instance.</td>
<td>54</td>
</tr>
<tr>
<td>ResourceContext getContext() Returns the resource context that this monitor belongs to</td>
<td>52</td>
</tr>
<tr>
<td>long getMonitoredPeriod() Returns the time period for which the usage of this resource type is monitored.</td>
<td>54</td>
</tr>
<tr>
<td>String getResourceType() The name of the resource type that this monitor represents</td>
<td>52</td>
</tr>
<tr>
<td>long getSamplingPeriod() Returns the sampling period for this resource type.</td>
<td>53</td>
</tr>
<tr>
<td>Comparable getUsage() Returns an object representing the current usage of this resource type by this resource context.</td>
<td>53</td>
</tr>
<tr>
<td>int hashCode() Retrieves the hashCode value of this ResourceMonitor.</td>
<td>54</td>
</tr>
<tr>
<td>boolean isDeleted() Returns true if the ResourceMonitor instance has been deleted, i.e. the delete() method has been called previously.</td>
<td>52</td>
</tr>
</tbody>
</table>
**Interface ResourceMonitor**

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>isEnabled()</td>
<td>Checks if the monitoring for this resource type is enabled for this resource context</td>
<td>52</td>
</tr>
<tr>
<td>void</td>
<td>notifyIncomingThread(Thread t)</td>
<td>Reports thread t is now attached to the ResourceContext associated to this monitor.</td>
<td>54</td>
</tr>
<tr>
<td>void</td>
<td>notifyOutgoingThread(Thread t)</td>
<td>Reports thread t is leaving the ResourceContext associated with this monitor.</td>
<td>54</td>
</tr>
</tbody>
</table>

**Method Detail**

**getContext**

```java
RESOURCECONTEXT getContext()
```

Returns the resource context that this monitor belongs to

**Returns:**
The associated ResourceContext

**getResourceType**

```java
STRING getResourceType()
```

The name of the resource type that this monitor represents

**Returns:**
The name of the monitored resource type

**delete**

```java
VOID delete()
```

Disable and delete this instance of Resource Monitor. This method MUST update the list of ResourceMonitor instances hold by the Resource Context (getContext().removeMonitor(this)).

**isEnabled**

```java
BOOLEAN isEnabled()
```

Checks if the monitoring for this resource type is enabled for this resource context

**Returns:**
true if monitoring for this resource type is enabled for this context, false otherwise

**isDeleted**

```java
BOOLEAN isDeleted()
```

Returns true if the ResourceMonitor instance has been deleted, i.e. the delete() method has been called previously.
Interface ResourceMonitor

Returns:
true if deleted.

enable

void enable() throws ResourceMonitorException, IllegalStateException

Enable the monitoring of this resource type for the resource context associated with this monitor instance. This method SHOULD also update the current resource consumption value (to take into account all previous resource allocations and releases occurred during the time the monitor was disabled).

Throws:
ResourceMonitorException - if the ResourceMonitor instance can not be enabled (for example some MemoryMonitor implementations evaluate the memory consumption by tracking memory allocation operation at runtime. This kind of Monitor can not get instantaneous memory value. Such Monitor instances need to be enabled at starting time.)
IllegalStateException - if the ResourceMonitor instance has been previously deleted

disable

void disable() throws IllegalStateException

Disable the monitoring of this resource type for the resource context associated with this monitor instance. The resource usage is not available until it is enabled again.

Throws:
IllegalStateException - if the ResourceMonitor instance has been previously deleted

getUsage

Comparable getUsage() throws IllegalStateException

Returns an object representing the current usage of this resource type by this resource context.

Returns:
The current usage of this resource type.

Throws:
IllegalStateException - if the ResourceMonitor instance is not enabled.

getSamplingPeriod

long getSamplingPeriod()

Returns the sampling period for this resource type.

Returns:
The sampling period in milliseconds, or -1 if a sampling period is not relevant for this resource type.
Interface ResourceMonitor

getMonitoredPeriod

long getMonitoredPeriod()

Returns the time period for which the usage of this resource type is monitored.

Returns:
The monitored period in milliseconds, or -1 if a monitored period is not relevant for this resource type.

notifyIncomingThread

void notifyIncomingThread(Thread t)

Reports thread t is now attached to the ResourceContext associated to this monitor.

Parameters:
t - the newly attached thread

notifyOutgoingThread

void notifyOutgoingThread(Thread t)

Reports thread t is leaving the ResourceContext associated with this monitor.

Parameters:
t - leaving thread

equals

boolean equals(Object resourceMonitor)

Checks if resourceMonitor is equals to the current instance. A ResourceMonitor rm1 is equals to a ResourceMonitor rm2 if rm1.getContext().equals(rm2.getContext()) and rm1.getType().equals(rm2.getType()).

Overrides:
equals in class Object

Returns:
true if the current instance is equals to the provided resourceMonitor

hashCode

int hashCode()

Retrieves the hashCode value of this ResourceMonitor. The hashCode value is based on the hashCode value of the associated ResourceContext and the hashCode value of the type.

Overrides:
hashCode in class Object

Returns:
hashCode
Class ResourceMonitorException

org.osgi.service.resourcemanagement

java.lang.Object
   java.lang.Throwable
      java.lang.Exception
         org.osgi.service.resourcemanagement.ResourceMonitorException

All Implemented Interfaces:
   Serializable

public class ResourceMonitorException
extends Exception

Resource Monitor Exception reports an invalid usage of a monitor.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceMonitorException(String msg)</td>
<td>55</td>
</tr>
</tbody>
</table>

Constructor Detail

public ResourceMonitorException(String msg)

Create a new ResourceMonitorException

Parameters:
- msg - message
Interface ResourceMonitorFactory

public interface ResourceMonitorFactory

A ResourceMonitorFactory are OSGI services used to create ResourceMonitor instance. These factories should only be used by ResourceManager singleton or authorities.

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String RESOURCE_TYPE_PROPERTY</td>
<td>56</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceMonitor createResourceMonitor(ResourceContext resourceContext)</td>
<td>56</td>
</tr>
<tr>
<td>String getType()</td>
<td>56</td>
</tr>
</tbody>
</table>

Field Detail

RESOURCE_TYPE_PROPERTY

public static final String RESOURCE_TYPE_PROPERTY = "org.osgi.resourcemanagement.ResourceType"

Method Detail

gType

String getType()

    Return the type of ResourceMonitor instance this factory is able to create.

    Returns:
        factory type

createResourceMonitor

ResourceMonitor createResourceMonitor(ResourceContext resourceContext)

    Create a new ResourceMonitor instance. This instance is associated with the ResourceContext instance provided as argument (ResourceContext.addResourceMonitor(ResourceMonitor) is called by the factory). The newly ResourceMonitor instance is disabled. It can be enabled by calling ResourceMonitor.enable().
Interface ResourceMonitorFactory

Parameters:
  resourceContext - ResourceContext instance associated with the newly created ResourceMonitor instance

Returns:
a ResourceMonitor instance

Throws:
  ResourceMonitorException
Class ThresholdException

org.osgi.service.resourcemanagement

java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ org.osgi.service.resourcemanagement.ThresholdException

All Implemented Interfaces:
  Serializable

class ThresholdException
  extends Exception

Threshold exception are thrown an invalid threshold setting is detected.

Constructor Summary

ThresholdException()  public class ThresholdException
                        extends Exception

ThresholdException

public ThresholdException()
### Interface Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPUMonitor</strong></td>
<td>A ResourceMonitor for the ResourceManager.RES_TYPE_CPU resource type.</td>
<td>60</td>
</tr>
<tr>
<td><strong>DiskStorageMonitor</strong></td>
<td>A ResourceMonitor for the ResourceManager.RES_TYPE_DISK_STORAGE resource type.</td>
<td>61</td>
</tr>
<tr>
<td><strong>MemoryMonitor</strong></td>
<td>A ResourceMonitor for the ResourceManager.RES_TYPE_MEMORY resource type.</td>
<td>62</td>
</tr>
<tr>
<td><strong>SocketMonitor</strong></td>
<td>A ResourceMonitor for the ResourceManager.RES_TYPE_SOCKET resource type.</td>
<td>63</td>
</tr>
<tr>
<td><strong>ThreadMonitor</strong></td>
<td>A ResourceMonitor for the ResourceManager.RES_TYPE_THREADS resource type.</td>
<td>64</td>
</tr>
</tbody>
</table>
Interface CPUMonitor

org.osgi.service.resourcemanagement.monitor

All Superinterfaces:
   ResourceMonitor

public interface CPUMonitor
extends ResourceMonitor

A ResourceMonitor for the ResourceManager.RES_TYPE_CPU resource type. CPUMonitor instance monitors the CPU consumed by a ResourceContext instance.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCPUUsage()</td>
<td>int</td>
<td>Returns the CPU usage as a cumulative number of nanoseconds. The ResourceMonitor.getUsage() method returns the same value, wrapped in a Long.</td>
</tr>
</tbody>
</table>

Methods inherited from interface org.osgi.service.resourcemanagement.ResourceMonitor

delete, disable, enable, equals, getContext, getMonitoredPeriod, getResourceType, getSamplingPeriod, getUsage, hashCode, isDeleted, isEnabled, notifyIncomingThread, notifyOutgoingThread

Method Detail

getCPUUsage

int getCPUUsage()

Returns the CPU usage as a cumulative number of nanoseconds.

The ResourceMonitor.getUsage() method returns the same value, wrapped in a Long.

Returns:

the CPU usage in nanoseconds
Interface DiskStorageMonitor

org.osgi.service.resourcemanagement.monitor

All Superinterfaces:
   ResourceMonitor

public interface DiskStorageMonitor
   extends ResourceMonitor


Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long getUsedDiskStorage()</td>
<td>Returns the sum of the size of the persistent storage areas of the bundles in this resource context.</td>
</tr>
</tbody>
</table>

Methods inherited from interface org.osgi.service.resourcemanagement.ResourceMonitor:
delete, disable, enable, equals, getContext, getMonitoredPeriod, getResourceType, getSamplingPeriod, getUsage, hashCode, isDeleted, isEnabled, notifyIncomingThread, notifyOutgoingThread

Method Detail

getUsedDiskStorage

long getUsedDiskStorage()

Returns the sum of the size of the persistent storage areas of the bundles in this resource context.

The ResourceMonitor.getUsage() method returns the same value, wrapped in a Long.

Returns:
the sum of the sizes of the persistent storage areas in bytes
**Interface MemoryMonitor**

`org.osgi.service.resourcemangement.monitor`

**All Superinterfaces:**

`ResourceMonitor`

```java
public interface MemoryMonitor
extends ResourceMonitor
```


### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>long getUsedMemory()</code></td>
<td>Returns the size of the java heap used by the bundles in this resource context.</td>
</tr>
</tbody>
</table>

### Method Detail

**getUsedMemory**

```java
long getUsedMemory()
```

Returns the size of the java heap used by the bundles in this resource context.

The `ResourceMonitor.getUsage()` method returns the same value, wrapped in a `Long`.

**Returns:**

the size of the used java heap in bytes
Interface SocketMonitor

org.osgi.service.resourcemangement.monitor

All Superinterfaces:
   ResourceMonitor

public interface SocketMonitor
extends ResourceMonitor

A ResourceMonitor for the ResourceManager.RES_TYPE_SOCKET resource type. SocketMonitor instance are used to monitor and limit the number of in-use sockets per ResourceContext instance. SocketMonitor instance handle all types of sockets (TCP, UDP, ...).

A TCP socket is considered to be in-use when it is bound (Socket.bind(java.net.SocketAddress)) or when it is connected (Socket.connect(java.net.SocketAddress)). It leaves the in-use state when the socket is closed (Socket.close()).

A UDP socket is in-use when it is bound (DatagramSocket.bind(java.net.SocketAddress)) or connected (DatagramSocket.connect(java.net.SocketAddress)). A UDP Socket leaves the in-use state when it is closed (DatagramSocket.close()).

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>long getSocketUsage()</td>
<td>Returns the number of existing socket created by a ResourceContext.</td>
</tr>
</tbody>
</table>

Methods inherited from interface org.osgi.service.resourcemangement.ResourceMonitor

delete, disable, enable, equals, getContext, getMonitoredPeriod, getResourceType, getSamplingPeriod, getUsage, hashCode, isDeleted, isEnabled, notifyIncomingThread, notifyOutgoingThread

Method Detail

getSocketUsage

long getSocketUsage()

Returns the number of existing socket created by a ResourceContext.

The ResourceMonitor.getUsage() method returns the same value, wrapped in a Long

Returns:
the number of existing socket.
Interface ThreadMonitor

org.osgi.service.resource.management.monitor

All Superinterfaces:
   ResourceMonitor

public interface ThreadMonitor
   extends ResourceMonitor


Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getAliveThreads()</td>
<td>64</td>
</tr>
</tbody>
</table>

Returns the number of alive threads created by the bundles in this resource context. A Thread is considered to be alive when its java state is one of the following:

- RUNNABLE
- BLOCKED
- WAITING
- TIMED_WAITING

The ResourceMonitor.getUsage() method returns the same value, wrapped in a Integer.

Returns:
the number of alive threads created by this resource context
7 Considered Alternatives

Resource Manager inside the Core framework or in a bundle?

Some framework operations like Bundle.start() or Bundle.stop() requires a context switching to account resource usage in the context of the related bundle (instead of using the context of the caller of the method). These context switching have to be done automatically by the framework. As a consequence, the Resource Manager solution must be implemented inside the core framework. However, the Resource Manager may take advantage of the extension bundle mechanism.

Adapt pattern or OSGi service?

Each bundle is belonging to one specific resource context. So, the Bundle.adapt() method is an easy to use way to get access to its related ResourceContext object. It also avoid service management code necessary to require and release a service reference.

Moreover, as the Core framework has to be modified in order to perform automatic context switching on specific framework methods, the adapt pattern is definitely the best approach.

03/22/2013: Evgeni (Prosyst) seems not to be 100% sure of the adapt pattern. He indicates that the service approach could be better. Then the discussion moves the reasons why the Start Level Service uses the adapt pattern.

08/20/2013: David Bosschaert proposes to remove the adapt pattern for retrieving Resource Manager and Resource Contexts. From the Enterprise Expert Group point of view, the Resource Manager capability may be implemented outside the OSGi core framework. Introducing the adapt pattern supposes to be a part of the OSGi core framework and implies a high coupling between the Resource Management solution and the core framework. David proposes to use the service mechanism to make available the Resource Manager instance. Then from this Resource Manager service, Resource Context can easily be retrieved.

Eventing paradigms

03/22/2013: Several eventing mechanism have been discussed:

- The Event Admin service. This is the logical service to send notifications on OSGi platform. However, this service is optional and may not be accessible all the time. What to do in these cases?

- Use of the core event system:
  - Notifications through BundleEvent objects. Notifications MUST be sent when a Resource Context instance exceeds one of its resource usage thresholds. As a Resource Context is not limited to a single bundle (it could be a set of bundle), the BundleEvent approach seems not to be a good approach.
  - Notifications through FrameworkEvent objects. Those events have been designed to notify about general events of the OSGi environment (e.g., The framework has started,...). FrameworkEvent instances contains a few fields like the bundle associated to the event, the exception causing this event and a type. Moreover, it is not possible to define filters when registering the FrameworkListener.
  - Notifications through a new ResourceEvent object. To be defined.

- Whiteboard pattern approach. Some ResourceListener instance (to be defined) may be registered as an OSGi service or any other pattern. Each time a ResourceMonitor instance detects a resource usage exceeding, it requests for the list of existing ResourceListener instances and notify them. Some filters may be applied in order to reduce the number of ResourceListener to be notified.

Resource event classes

04/03/2013: Resource Event instances notify applications about threshold exceeding as well as Resource Management configuration updates (e.g., adding/removing Resource Context instance, adding/removing bundles from Resource Context instance). A Resource Listener instance receiving a notification then has to identify which kind of notification it receives (through a call to ResourceEvent.getType()) and treat it accordingly.
So it could be interesting to separate concerns by creating two different Event/Listener interfaces. Resource Event and Resource Listener can still deal with threshold exceeding while Resource Context Event and Resource Context Listener will manage any events about Resource Context configuration updates (adding or removing Resource Context, adding or removing bundles).

**CPU Monitor**

CPU Monitor instances monitor the CPU usage of Resource Context. The CPU usage is expressed as a percentage of usage over the monitoring period.

This percentage may be evaluated using the raw cpu data like the number of nanoseconds a Resource Context uses CPU. CPU Monitor instance evaluates the percentage by making a difference between cumulative values periodically retrieved (sampling period).

It could be interested to leave access to the CPU cumulative values. A getCumulativeUsage() method may be added to the CPU Monitor interface.

**Resource Threshold algorithm and eventing**

When the resource consumption exceeds the WARNING threshold, the Resource Threshold instance goes from the NORMAL state to the WARNING state and generates WARNING Resource Event. When the resource consumption decreases below the WARNING threshold, the Resource Threshold instance goes back to the NORMAL state and generates a NORMAL Resource Event notification. In some cases, the resource consumption may fluctuate around the threshold generating several WARNING or NORMAL Resource Event notification. The next chart summarizes the situation:

NTT proposes the Resource Threshold instances are not a fixed straight value but rather a floating value with a lower warning threshold and a higher warning threshold (some kind of range). The NTT’s solution is summarized below:
WARNING Resource event are generated only when higher warning threshold is exceeded. The same for NORMAL Resource Event when the resource consumption decreases under the lower warning threshold.

This kind of algorithms decreases the number of Resource Event notifications but increases the implementation complexity.

The RFC does not take any strong position on that particular question. Resource Management solution providers are free to implement such algorithms.

**Resource Monitor Factory**

ResourceMonitor instances are created by ResourceMonitorFactory. Some ResourceMonitor implementation (e.g., those provided by the framework itself) might not have ResourceMonitorFactory. This is strongly encouraged to provide Resource Monitor Factory instance for all kind of Resource Monitor.

**Resource accounting and context switching**

In a previous version, the Resource Manager service also allowed to perform Resource Context switching. The switching feature relied on a specific design for resource monitoring and accounting. For generality, it was decided to remove this feature (after CPEG review).

This feature relies on the association of Java threads with a Resource Context instance. In that case, every Java thread is associated with a single Resource Context instance. A Resource Context switching operation then reallocates a Java thread to another Resource Context instance. Therefore, all further resource allocation made during the execution of the Java thread are accounted to the new Resource Context instance.

This switching feature is executed by a call to the switchContext(Thread, ResourceContext) or switchCurrentContext(ResourceContext). Resource Monitors are notified when a context switching operation occurs. When a thread joins a ResourceContext, all ResourceMonitors of the incoming ResourceContext are notified through a call to ResourceMonitor.notifyIncomingThread(thread). On the opposite, when a thread leaves a ResourceContext, all ResourceMonitors of the outgoing ResourceContext are notified through a call to ResourceMonitor.notifyOutgoingThread(thread).

Below are described three use cases showing an appropriate usage of the context switching feature.

When the platform is starting, the core framework may start automatically bundles. The next sequence diagram describes the actions performed to start a bundle:
For every bundle instance to be started, the core framework calls `Bundle.start()` method (1). `Bundle.start()` method then creates a new instance of the BundleActivator implementation class of the Bundle (2) and calls `BundleActivator.start()` method (3). `BundleActivator.start()` method is generally used to allocate all resources a bundle needs, start threads and get and/or register services.

The object starting the bundle (actually the thread calling `Bundle.start()` method) belongs to the System Resource Context. As a consequence, resources consumed during start() call, including the resources used for the activation of the bundle, are accounted by the System Resource Context.

This situation may not be suitable. In usual situations, these resources may be accounted by the Resource Context instance of the bundle. A context switching operation has thus to be performed to switch to the Resource context instance of the bundle. The next diagram summarizes the actions to perform:
The ResourceManager.switchContext() operation (action 2) switches to the Resource Context of the bundle. At this point, all resources consumed by the current thread are accounted by the Resource Context instance of the bundle. Then, a new BundleActivator instance is created (action 3) and the BundleActivator.start() method is called (action 4). BundleActivator.start() method allocates all resources needed by the bundle. Before the completion of the Bundle.start() method, a context operation is again executed to switch back to the System Resource Context (action 5).

It is important to note that a context switching operation must be executed twice. The first context switching sets the context to which resources must be accounted. The second switches back to the initial context. The developer MUST ensure that these two operations are correctly achieved in order to avoid resource miscellaneous accounting.

This use case clearly shows that a context switching operation is needed at bundle startup to account resource in the appropriate Resource Context instance. Other framework operations requires a context switching execution:

- BundleActivator.start() - switch to the to-be-activated bundle Resource Context instance.
- BundleActivator.stop() - switch to the to-be-stopped bundle Resource Context instance.
- ServiceListener.serviceChanged() - switch to the Resource Context instance of the bundle hosting the ServiceListener instance.
- BundleListener.bundleChanged() - switch to the Resource Context instance of the bundle hosting the BundleListener instance.
- ServiceFactory.getService() - switch to the Resource Context instance of the bundle requesting the OSGi service.

Some other Compendium OSGi service like Http Service or Event Admin service should also take advantage of the context switching operation. The next diagram shows the use case for the Event Admin service:
The EventAdmin Service should execute a context switching operation before and after calling EventHandler.handleEvent(event). This operation switches the current context to the context of the bundle providing the Event handler. Then, the resources consumed by the EventHandler.handleEvent() method are accounted by the Resource Context associated to the bundle providing the Event Handler and are not paid by the Event Admin service.

Now, the LogService case is considered. Any bundle about to log requires the LogService instance and calls LogService.log() method. The next schema shows two bundles A and B belonging respectively to Resource Context RCa and RCb. These two bundles requires the LogService (which belongs to Resource Context Log Service):

In such situation, the resources used by the LogService.log() method due to the bundle A are paid by RCa (the Resource Context of A). So, if bundle A overuses the LogService, it may exceeds the resource quotas of RCa which may lead to stop the bundle A (for example). Bundle B is not impacted and can still use the LogService.

Now, the following example is considered:
Bundle A belongs to the Resource Context named RCa. It requires the Red service provided by Bundle B. The Red service provides a single public method called m(). Red.m() method uses the LogService to log useful data. Bundle B belongs to Resource Context RCb, LogService bundle belongs to Resource Context Log Service.

When bundle A calls Red.m() method, all the resources consumed by Red.m() method are paid by RCa (if, for instance, the owner is defined as the owner of the current thread). As Red.m() calls also the LogService.log() method, the resource consumed by LogService.log() method are also paid by RCa.

Now, if the Red.m() method executes a context switching operation to switch to RCb (resource context instance of bundle B), then all resources used by Red.m() method are paid by RCb including the resources used by the call to the LogService.log() method. In this case, a call to Red.m() method costs nothing to RCa. However, Red.m() method MUST switches back to RCa at the end of its execution. If it is not the case, all resources consumed afterwards could be still paid by RCb. One more time, it is very important for a method that need to execute a switch context operation to switch back to the initial context at the end of its execution.

The three use cases described above show that context switching operation are not suitable in all cases. The most important thing is finally to determine which is the context executing the code. In all cases, a switching context operation notify all ResourceMonitor instances of both the outgoing ResourceContext and the incoming ResourceContext. The ResourceMonitor instances of the leaving ResourceContext are notified through ResourceMonitor.notifyOutgoingThread(thread) where the ones of the incoming ResourceContext are notified through ResourceMonitor.notifyIncomingThread(thread).

Compatibility between bundles capabilities

Some bundle implementation should naturally use the Resource Management features and in particular the switching context operation in order to account resource properly. For example, the Http Service implementation switch to the Context of the bundle providing the servlet before executing any service methods when receiving an HTTP request.

An open issue is the deployment of resource management non-aware bundles into a resource management aware context. In that particular case, some Resource Context may not be charged correctly.

The present specification gives all the features necessary to implement bundles and framework resource management aware. As a consequence, the platform operator is free to deploy suitable bundles depending on its needs (i.e., in a resource management aware context or not).

Implementation of resource management aware bundles

There are different ways to implement a resource management aware bundle.

The first approach is the direct implementation into the bundle. The bundle implementer is free to execute context switching operation when it is required. The bundle implementer is fully responsible to choose the appropriate Resource Context to charge and it is also responsible to switch back to the initial Resource Context. This approach has some drawbacks. First, it implies to handle Resource Management features into a business features which is sometimes not suitable. Moreover, the platform operator must be confident in the way the bundle implementer uses...
context switching operations (i.e., permissions). Finally, as the context switching operations are made by the bundle itself, the chosen accounting policy is statically defined. This accounting policy may not be convenient in all situations.

The second approach is to take advantage of the weaving feature. The weaving feature allows to inject some bytecode into existing bundles. In this case, the injected byte-code deals with the context switching features. The byte-code injection can be done at runtime or at the compilation time. The advantage of this approach is that the bundle implementer does not need to handle any context switching feature. Moreover, the accounting policy can be changed with a new byte-code injection. The main drawback of this approach is the complexity and acceptability of the use of bytecode injection.

Finally, the last approach is to implement service proxies. This kind of service proxyfies a service which is not resource management aware. The service proxy implements the same interface as the one of the to-be-proxified service and handles all the resource management stuff. This approach is very similar to the weaving approach.

Example 1
Operators involved into the Residential market plan to provide a Resource Management aware OSGi platform on which Third Party Players will deploy bundles:

The OSGi platform hosts some operator bundles like HttpService or LogService. All of them are resource management aware, i.e., they automatically execute switch context operation in order to charge resource to the appropriate Resource Context.

All these operator services are used by Third Party Player bundles. Each Third Party Player is isolated from another (they do not share services between them) and the platform operator creates a Resource Context per Third Party Player. Moreover, as Third Party Players may not be trusted, their bundles do not have access to the Resource Management API. This does not prevent the resources to be accounted to the Resource Context belonging to the Third Party Player.

In such situations, Resource Management is transparent for the Third Party Player. Authorities and operator bundles handle resource context switching at the appropriate moment.

Example 2
In the Enterprise context, bundles are deployed using the OBR service. The OBR allows to deploy bundles and all their dependencies computed based on package and service dependencies of to be-installed bundle:
Here, the OBR service is requested to install the bundle 1 which depends on a package provided by bundle 2 (which is not installed on the platform). The OBR then installs the bundle 1 as well as the bundle 2 in order to satisfy the missing dependency.

In the case where the OSGi Core platform is resource management aware, the operator may have to deploy bundles which should be resource management aware. So the OBR service should take care between a bundle implementation which is resource management aware (light orange box on the schema) and a one which is not (dark orange box).

In such cases, either the OBR service is smart to deploy the appropriate version or the operator should do it itself.

**Event Admin use-case**

The next sequence diagram shows the use-case of the Event Admin:
A bundle uses the EventAdmin.sendEvent() method to emit an event. The EventAdmin service then performs a lookup into the OSGi service registry to find out all the available EventHandler services. For every to-be-notified EventHandler, the EventAdmin performs a switching context operation to be in the Resource Context of the bundle providing the EventHandler service. Then it calls the EventHandler.handleEvent() method and switches back to the initial Resource Context (the one of the bundle sending the event).

The context switching operation made here allows to charge the bundle receiving the event instead of the bundle sending the event.

**Http Service use-case**

The Http Service use-case is shown below:
A browser requests an HTTP GET on context/index.html. The HttpService identifies the HttpServlet to be-invoked based on the context of the request. Once identified, the HttpService executes a switch context operation to the Resource Context of the bundle provided the HttpServlet. Then, the HttpService calls the HttpServlet.doGet() method and switches back to the initial context.

The switching context operation allows to charge the bundle providing the HttpServlet instead the bundle exposing the HttpService.

# Security Considerations

# Document Support

## References

3. RFP 153 – Resource Management for OSGi Platform, OSGi Alliance
5. Y. Maurel, A. Bottaro, R. Kopetz, K. Attuchi. Adaptive Monitoring of End-user OSGi based Home


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.

Author's Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Company</th>
<th>Address</th>
<th>Voice</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>BONNARDEL Gregory</td>
<td>France Telecom Orange</td>
<td>28 Chemin du Vieux Chêne, 38240 Meylan, France</td>
<td>+33 4 76 76 44 49</td>
<td><a href="mailto:gbonnardel.ext@orange.com">gbonnardel.ext@orange.com</a></td>
</tr>
<tr>
<td>BOTTARO Andre</td>
<td>France Telecom Orange</td>
<td>28 Chemin du Vieux Chêne, 38240 Meylan, France</td>
<td>+33 4 76 76 41 03</td>
<td><a href="mailto:andre.bottaro@orange.com">andre.bottaro@orange.com</a></td>
</tr>
<tr>
<td>DIMOV Svetozar</td>
<td>Prosyst</td>
<td></td>
<td></td>
<td><a href="mailto:s.dimov@prosyst.com">s.dimov@prosyst.com</a></td>
</tr>
<tr>
<td>GRIGOROV Evgeni</td>
<td>Prosyst</td>
<td></td>
<td></td>
<td><a href="mailto:e.grigorov@prosyst.com">e.grigorov@prosyst.com</a></td>
</tr>
<tr>
<td>Name</td>
<td>RINQUIN Arnaud</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Company</td>
<td>France Telecom Orange</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>28 Chemin du Vieux Chêne, 38240 Meylan, France</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voice</td>
<td>+33 4 76 76 45 59</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:arnaud.rinquin@orange.com">arnaud.rinquin@orange.com</a></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>CHAZALET Antonin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>France Telecom Orange</td>
</tr>
<tr>
<td>Address</td>
<td>28 Chemin du Vieux Chêne, 38240 Meylan, France</td>
</tr>
<tr>
<td>Voice</td>
<td>+33 4 76 76 41 03</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:antonin.chazalet@orange.com">antonin.chazalet@orange.com</a></td>
</tr>
</tbody>
</table>

**Acronyms and Abbreviations**

**End of Document**
Abstract

This RFC proposes various updates to the Subsystem Service Specification.
0  Document Information

0.1  License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0 Document Information..................................................................................................................2
  0.1 License.................................................................................................................................2
  0.2 Trademarks..........................................................................................................................3
  0.3 Feedback..............................................................................................................................3
  0.4 Table of Contents................................................................................................................3
  0.5 Terminology and Document Conventions.................................................................4
  0.6 Revision History...................................................................................................................4

1 Introduction.................................................................................................................................6

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

3 Problem Description..................................................................................................................7
  3.1 Bug 2251 – Translation of Subsystem Headers.................................................................7
  3.2 Bug 2430 - Subsystems spec needs to account for Weaving Service downstream............7
  3.3 Bug 2270 - Subsystems should allow a Deployment.MF to be provided separately.........8
  3.4 Bug 2211 – Subsystem API to provide access to more information.................................8
  3.5 Bug 2081 - Determining service dependencies for application subsystems..................8
  3.6 Bug 2537 - Preferred-Provider not always a cure for uses constraint violation................8

4 Requirements.............................................................................................................................9

5 Technical Solution.....................................................................................................................9
  5.1 Translation of Subsystem Headers.......................................................................................9
    5.1.1 Subsystem-Localization Header...................................................................................9
    5.1.2 Localization Properties...............................................................................................10
5.1.3 Locating localization entries................................................................. 10
5.1.4 Informational Subsystem Headers....................................................... 10
5.2 Weaving Hooks......................................................................................... 11
5.3 Allow Deployment Manifest to be Provided Separately............................ 12
5.4 Subsystem API to provide access to more information............................. 12
5.5 Application Subsystem Service Dependencies......................................... 12
5.6 Bug 2517 - Preferred-Provider not always a cure for uses constraint violation ........................................... 12
  5.6.1 Nothing.............................................................................................. 13
  5.6.2 Content + Preferred Provider Repository............................................ 13
  5.6.3 Content + Preferred Provider + System Repository............................ 13
  5.6.4 Content + Preferred Provider + System + Local Repository............. 13
  5.6.5 All Repositories.................................................................................. 13

6 Considered Alternatives................................................................................... 13
  6.1 Weaving Hooks and DynamicImport-Package........................................ 14
  6.2 Subsystem API to provide more access to information............................ 14
  6.3 Bug 1916 – Define the TCCL for Scoped Subsystems............................... 14
  6.4 TCCL Reliance (from Application Domain section).................................... 14
  6.5 Subsystem Configuration........................................................................ 15
    6.5.1 Requirements.................................................................................. 15
    6.5.2 Problem Description....................................................................... 15
    6.5.3 Technical Solution.......................................................................... 15
    6.5.4 Delivering Configuration Data......................................................... 16
    6.5.5 Configuration Admin Service Visibility............................................ 17
  6.6 Configuration Resources......................................................................... 17
  6.7 Manifest matching rules.......................................................................... 17

7 Security Considerations.................................................................................... 18

8 Javadoc........................................................................................................... 18

9 Document Support.......................................................................................... 44
  9.1 References.............................................................................................. 44
  9.2 Author's Address.................................................................................... 44
  9.3 Acronyms and Abbreviations.................................................................... 44
  9.4 End of Document..................................................................................... 44

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

0.6 Revision History

The last named individual in this history is currently responsible for this document.
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Initial  | 9th April 2013| Created  
Graham Charters, IBM (charters@uk.ibm.com)                                                                                                                                 |
| 0.1      | 7th June 2013 | Updates to domain, problem desc, service dep calc, etc, based on last telecon review.  
Graham Charters, IBM (charters@uk.ibm.com)                                                                                                                                 |
| 0.2      | 7th June 2013 | Added sections 5.5 – 5.7.  
John Ross, IBM (jwross@us.ibm.com)                                                                                                                                 |
| 0.3      | 7th June 2013 | Added Configuration section  
Tom Watson, IBM (tjwatson@us.ibm.com)                                                                                                                                 |
| 0.4      | 17th June 2013| Accepted all changes as of the Jun 13 OSGi F2F.  
Updated sections 3.2 and 5.7 through 5.7 based on F2F feedback.  
John Ross, IBM (jwross@us.ibm.com)                                                                                                                                 |
| 0.5      | 27th June 2013| Accepted all changes reviewed on the 26th June 2013 EEG call.  
Added the text of the comment in 5.5 regarding outstanding issues with woven dynamic imports becoming part of the sharing policy into the document. Deleted the comment.  
Added the text of the comment in 5.5 regarding why DynamicImportPackage was not addressed to the Considered Alternatives chapter. Deleted the comment.  
Added the text of the comment in 5.7 regarding SubsystemDTO to the Considered Alternatives section. Deleted the comment.  
Added section 3.10 based on bug 2537.  
John Ross, IBM (jwross@us.ibm.com)                                                                                                                                 |
| 0.6      | 29th October 2013| Minor edits.  
Created javadoc and linked to RFC.  
Updated table of contents.  
John Ross, IBM (jwross@us.ibm.com)                                                                                                                                 |
| 0.7      | 30th October 2013| Add abstract.  
Add descriptions to information subsystem headers.  
Update javadoc with informational subsystem headers.  
John Ross, IBM (jwross@us.ibm.com)                                                                                                                                 |
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>5th November 2013</td>
<td>Minor edits. Delete all information associated with handling the DynamicImport-Package header (requirement SSU025). Move to Considered Alternatives. Delete all information associated with the TCCL (requirement SSU030, sections 2.1, 3.3, and 5.3). Move to Considered Alternatives. Delete all information associated with Subsystem Configuration (requirements SSU070 and SSU075, sections 3.7, 5.8, and 5.9). Move to Considered Alternatives. Delete all information associated with Manifest Matching Rules (section 3.8). Move to Considered Alternatives. Delete comments associated with the previous topics. Move to Considered Alternatives. John Ross, IBM (<a href="mailto:jwross@us.ibm.com">jwross@us.ibm.com</a>)</td>
</tr>
<tr>
<td>0.9</td>
<td>14th November 2013</td>
<td>Accept all changes discussed at November F2F. Reformatting and reorganization with no content changes (untracked). Update revision history. Remove section 3.6. Remove requirement SSU026. Add description of syntax requirements for informational headers to section 5.1.4. Add chosen solution for bug 2537 to section 5.6. Add security considerations. Add authors. John Ross, IBM (<a href="mailto:jwross@us.ibm.com">jwross@us.ibm.com</a>)</td>
</tr>
<tr>
<td>1</td>
<td>21st November 2013</td>
<td>Accept all changes discussed on November 20 EEG call. Delete remaining template text. Fix typo in requirement SSU080. Remove Address and Voice rows from author tables. John Ross, IBM (<a href="mailto:jwross@us.ibm.com">jwross@us.ibm.com</a>)</td>
</tr>
</tbody>
</table>

### 1 Introduction

The Subsystems specification defines how to describe and provision bundle collections into an OSGi framework. Subsystem types have different sharing policies enabling a variety of use cases (e.g. isolated applications running in an application server, or non-isolated features used to assemble a middleware product).
It was not possible to satisfy all requirements in the first version of the specification and as Subsystems has been used in the wild, new requirements have also come to light. This RFC addresses a prioritized set of requirements in the form of a minor update to the Subsystems specification.

2 Application Domain

The application domain for this RFC is the use of Subsystem 1.0. Further background can be found in the application domain of the subsystem 1.0 RFC (RFC 152). Subsystem 1.0 was first standardized in OSGi Enterprise R5. A few requirements that were known at the time were not addressed due to specification schedules, and since the release of R5, user experience has resulted in new requirements being raised. This RFC is intended to gather these requirements together into a 1.1 update to the Subsystem specification.

3 Problem Description

3.1 Bug 2251 – Translation of Subsystem Headers

Products that support Subsystem for applications or runtime features may have tools that display information about available or installed Subsystems, and so it should be possible to translate the human readable headers (e.g. Subsystem-Name, Subsystem-Description).

In addition to translating the existing headers, we should consider adding more of the informational headers that are available to bundles, such as Bundle-License.

3.2 Bug 2430 - Subsystems spec needs to account for Weaving Service

The Core R5 spec section 56, "Weaving Hook Service Specification" defines a whiteboard service by which weaving hooks can introduce new package dependencies into java byte code as it is loaded. Section 56.3 states, "These dynamically added dependencies are made visible through the Bundle Wiring API Specification on page 133 as new requirements."

The Enterprise R5 spec section 134 introduces the "Subsystem Service Specification." Section 134 does not explicitly discuss how new requirements introduced by weaving hooks should be handled. Woven requirements are introduced at run time, often after a deployment has been precalculated.

Overall, a design is required that handles dependencies introduced dynamically via weaving hooks. What we need out of this bug is a standard mechanism by which a Scoped Subsystem's resolver hooks can detect and permit dynamic requirements to be allowed into that subsystem.
Section 134.6, "Determining dependencies" should explicitly address how dynamically introduced dependencies should be handled.

Section 134.15.5 should state how a Scoped Subsystem's associated Region's sharing policy will permit packages matching dynamically introduced dependencies to be allowed into the Scoped Subsystem.

Both sections 134.16.2.1 and 134.16.3.2 should be updated to account for dynamic (woven) requirements.

See RFC 191 – Weaving Hook Enhancements - this was created as a pre-requisite for enabling weavers to enhance bundles inside scoped subsystems.

### 3.3 Bug 2270 - Subsystems should allow a Deployment.MF to be provided separately

As a Subsystem moves through the software life-cycle, from development, through QA testing and into production, it is often necessary to be able to lock down the deployment to ensure what goes into production is exactly what was tested. A deployment is locked down using a deployment manifest which must be included in the subsystem archive (.esa). However, a deployment is likely to be locked down after the initial subsystem archive has been created and so in some circumstances it is desirable to not have to crack open the archive file in order to put the deployment manifest in. Instead it should be possible to install a subsystem and provide its deployment manifest separately in the same installation step.

### 3.4 Bug 2211 – Subsystem API to provide access to more information

Subsystems 1.0 only provides very limited access to subsystem information through the API. At a minimum, we need to provide access to the deployment manifest as this is potentially generated by the Subsystem runtime and a management agent may need it to reason about a deployed subsystem.

### 3.5 Bug 2081 - Determining service dependencies for application subsystems

Subsystems 1.0 requires scoped subsystem resolution to prefer services provided by the contents of the subsystem. It states that the services provided and required by bundles may be derived from DS or Blueprint, but does not provide a normative approach to declaring services that is portable and component model agnostic.

Enterprise OSGi R5 defines the osgi.service namespace for declaring service capabilities and requirements. This mechanism could be standardized as a component model agnostic complement to the use of DS and Blueprint for determining service dependencies during subsystem provisioning.

needs more spec clarification. Main thing is to use the osgi.service capability with an effective attribute other than 'resolve'. How the capability/requirement is added to the bundle can be left to a tool or can be done directly by the developer. In DS/Blueprint cases this metadata can be inferred from the xml file, but how thats done is left up to a tool.

### 3.6 Bug 2537 - Preferred-Provider not always a cure for uses constraint violation.

The issue is highlighted by the following simplified scenario.

- Root Subsystem
  - Bundle A - Export-Package: foo,bar;uses:=foo
  - Child Subsystem
    - Bundle B - Export-Package: foo
Based on the current specification, a uses constraint violation will always result from this scenario, even if Bundle A is listed as a Preferred-Provider. Content resources are strictly favored (i.e. if a matching capability is found in the Content Repository, the search must stop, so the resolver is not given Bundle A as an option) so Bundle C will always get wired to Bundle B for package foo. Since the only provider of package bar is Bundle A, a uses constraint violation results.

Note that the solution, if any, could not simply be to reverse the current Content followed by Preferred Provider repository lookup. If Preferred Provider came first, you'd encounter the same issue by switching the positions of Bundle A and Bundle B. The solution would need to include the capabilities from both the Content and Preferred Provider repositories in the list of capabilities returned to the resolver.

4 Requirements

SSU010 – The specification MUST make it possible to provide translations for all human readable Subsystem-Headers and package these inside a Subsystem archive file.

SSU015 – The specification MUST define a standard set of informational headers similar to those available for bundles (e.g. Bundle-License).

SSU020 – The specification MUST make it possible for a weaving hook to add new package imports to a scoped subsystem's sharing policy in order to allow resolution of any new package dependencies it has woven into a subsystem content bundle.

SSU040 – The specification MUST make it possible to install a subsystem through a subsystem archive and provide its deployment manifest separately, but as part of the same installation step.

SSU050 – The specification MUST make it possible to retrieve a subsystem's deployment manifest through the subsystem API. The deployment manifest MAY contain additional information over and above what was provided during installation.

SSU080 – The specification SHOULD make it possible to selectively access the services of an application subsystem from outside the application.
5 Technical Solution

5.1 Translation of Subsystem Headers

For consistency and ease of comprehension, the design for localizing subsystem manifest headers follows the approach used by bundles.

5.1.1 Subsystem-Localization Header

The subsystem localization header identifies the default base name of the localization properties files contained in the subsystem archive. The default value is OSGI-INF/l10n/subsystem. Translations are therefore, by default, OSGI-INF/l10n/subsystem_de.properties, OSGI-INF/l10n/subsystem_nl.properties, and so on. An example of the header that specifies the default is:

```
Subsystem-Localization:  OSGI-INF/l10n/subsystem
```

The location is relative to the root of the subsystem archive.

5.1.2 Localization Properties

A localization entry contains key/value entries for localized information. All headers in a subsystem’s manifest can be localized. However, the subsystems implementation must always use the non-localized versions of headers that have subsystem semantics.

A localization key can be specified as the value of a subsystem’s manifest header using the following syntax:

```
header-value ::= '%'text
text ::= < any value which is both a valid manifest header value and a valid property key name >
```

For example, consider the following subsystem manifest entries:

```
Subsystem-Name: %acme subsystem
Subsystem-Description: %acme description
Subsystem-SymbolicName: acme.Subsystem
Acme-Defined-Header: %acme special header
```

User-defined headers can also be localized. Spaces in the localization keys are explicitly allowed.

The previous example manifest entries could be localized by the following entries in the manifest localization entry OSGI-INF/l10n/subsystem.properties.

```
# subsystem.properties
acme\ subsystem=The ACME Subsystem
acme\ description=The ACME Subsystem provides all of the ACME \ services
acme\ special\ header=user-defined Acme Data
```

The above manifest entries could also have French localizations in the manifest localization entry OSGI-INF/l10n/subsystem_fr_FR.properties.

5.1.3 Locating localization entries

The Subsystems implementation must search for localization entries by appending suffixes to the localization base name according to a specified locale and finally appending the .properties suffix. If a translation is not found, the locale must be made more generic by first removing the variant, then the country and finally the language until...
an entry is found that contains a valid translation. For example, looking up a translation for the locale en_GB_welsh will search in the following order:

`OSGI-INF/l10n/subsystem_en_GB_welsh.properties`
`OSGI-INF/l10n/subsystem_en_GB.properties`
`OSGI-INF/l10n/subsystem_en.properties`
`OSGI-INF/l10n/subsystem.properties`

5.1.4 Informational Subsystem Headers

All of the following headers have bundle manifest analogs and will have the same syntax restrictions as described in section 3.2.1 of the OSGi Core Specification.

5.1.4.1 Subsystem-Category

Manifest header identifying the categories of the subsystem as a comma-delimited list.

5.1.4.2 Subsystem-Copyright

Manifest header identifying the subsystem's copyright information.

5.1.4.3 Subsystem-DocURL

Manifest header identifying the subsystem's documentation URL, from which further information about the subsystem may be obtained.

5.1.4.4 Subsystem-License

Manifest header identifying the subsystem's license.

5.1.4.5 Subsystem-Vendor

Manifest header identifying the subsystem's vendor.

5.1.4.6 Subsystem-ContactAddress

Manifest header identifying the contact address where problems with the subsystem may be reported; for example, an email address.

5.1.4.7 Subsystem-Icon

Manifest header identifying the icon URL for the subsystem.

5.2 Weaving Hooks

This section describes the technical solution for handling dynamic imports added by weaving hooks.

Dynamic package imports added by weaving hooks are observed by registering a WovenClassListener service and receiving notifications via WovenClassListener.modified(WovenClass). If the woven class is in the TRANSFORMED state [1], the bundle containing the woven class is obtained by calling WovenClass.getBundleWiring().getBundle(). The scoped subsystem, if any [2], containing the bundle as a constituent is retrieved. If necessary [3], the subsystem is resolved [4]. For each dynamic import, if necessary [5], the subsystem's sharing policy is updated [6].

[1] The sharing policy must be updated while the woven class is in the TRANSFORMED state so that it takes effect before the bundle wiring is updated during the transition to DEFINED; otherwise, the class would fail to load.
[2] A bundle might be a constituent of multiple subsystems, but never more than one scoped subsystem. The rest are features, which have no sharing policies to update. It's possible the bundle will not be a constituent of a scoped subsystem.

[3] It's possible for a classload request to occur on a bundle in an unresolved subsystem because the framework is free to resolve bundles whenever it desires. A resolved bundle can potentially receive a classload request. For example, a BundleEventListener registered with the system bundle context could receive the RESOLVED event and, for whatever reason, load a class. Also, a resolved bundle in an unresolved feature might get wired to another bundle.

[4] The subsystem must be resolved in order to guarantee the dynamic imports will not effect the resolution and, therefore, potentially create a wiring inconsistent with the deployment manifest.

[5] The sharing policy is only updated if the dynamic import cannot be completely satisfied from within the subsystem. Note that all dynamic imports with a wildcard must always be added to the sharing policy.

[6] We encounter the same issue here as in supporting the DIP header in terms of what happens when bundles are refreshed or a child subsystem is installed into an already resolved parent. It's possible that the runtime wiring will be inconsistent with the deployment manifest. We punt on specifying this under the assumptions that (1) weaving hooks are scant and (2) weaving hook providers really know what they're doing and will minimize or altogether avoid the use of wildcards in dynamic imports. We therefore consider any issues to be unlikely and rare.

5.3 Allow Deployment Manifest to be Provided Separately

A new method is added to the Subsystem interface with the following signature.

public Subsystem install(String location, InputStream content, InputStream deploymentManifest);

This method installs a subsystem using the provided deployment manifest instead of the one in the archive or the computed one. If the deployment manifest is null, the behavior is exactly the same as in the install(String, InputStream) method. Implementations must support input streams in the format described by section 134.2 of the Subsystem Service Specification. If the deployment manifest does not conform to the subsystem manifest (see 134.15.2), the installation fails and a SubsystemException is thrown. If the input stream throws an IOException, the installation fails and a SubsystemException is thrown with the IOException as the cause.

5.4 Subsystem API to provide access to more information

A new method is added to the Subsystem interface with the following signature.

public Map<String, String> getDeploymentHeaders();

The method follows the same rules as described for the getSubsystemHeaders(Locale) method except that no Locale is accepted since there are currently no translatable headers defined.

5.5 Application Subsystem Service Dependencies

Section 134.16.2.2 of the Subsystem 1.0 specification states the following regarding Service Imports:

"Application resolution is required to prefer services provided by content bundles over those provided outside the application. For this reason, the application Subsystem sharing policy only imports services required by the Subsystem's content bundles that are not also provided by the content bundles. There is no standard way to determine this, but a Subsystem runtime is permitted to use its own means to determine the services to import. Examples include resource metadata from a bundle repository, or analysis of bundle contents (e.g. Blueprint or Declarative Service configurations). A deployment manifest for an application Subsystem would list these service imports using the Subsystem-ImportService header."

This specification provides a means of declaratively identifying the services a bundle provides using the Provide-Capability and Require-Capability headers with the osgi.service namespace (as described in Enterprise OSGi R5, section 135.4).

An example of a bundle providing the service and declaring it using the Provide-Capability header is as follows:
Provide-Capability: osgi.service;
    objectClass=com.foo.MyService;
    a.service.prop=SomePropertyValue

An example of a bundle requiring a service and declaring the requirement using the Require-Capability header is as follows:

Require-Capability: osgi.service;
    filter="(&\(objectClass=com.foo.MyService\)(a.service.prop=SomePropertyValue))"

These headers can be hand-written, for example to declare programmatic use of an OSGi service, or can be generate by a tool, such as BND, based on declarative component model configuration, e.g. Declarative Services or Blueprint. A subsystem implementation must assume that if these headers are present in a bundle, they declare all the service dependencies of that bundle. The implementation must therefore not search the bundle for additional dependencies from other sources, such as contained Blueprint or DS XMLs.

5.6 Bug 2517 - Preferred-Provider not always a cure for uses constraint violation.

The following sections represent possible solutions to the issue.

5.6.1 Nothing

Do nothing. This is working as designed.

5.6.2 Content + Preferred Provider Repository

During the subsystem install process, while resolving the content of the subsystem (offline), change the steps outlined in section 134.6 - "Determining Dependences" to state that the search continues from step 1 "The Content Repository" to step 2 "The Preferred Repository" even if a capability is found in the content repository. The subsystem must prefer capabilities from the content repository, but should allow capabilities from the preferred repository if they can be used to solve an inconsistent class space (uses) issue.

5.6.3 Content + Preferred Provider + System Repository

While option 2 gives the developer an out to work around uses constraint issues when providers are available in the system repository, it still requires developer intervention to add a Preferred-Provider header. Also, preferred providers can only be used against providers that are contained in the parent subsystem (not any higher in the hierarchy). This may not always be possible since existing providers may pre-exist higher up in the hierarchy. An extension of option 2 is to also change step 2 "The Preferred Repository" to state that the search continues from step 2 "The Preferred Repository" to step 3 "The System Repository" if and only if no matching capabilities are found in the preferred repository. The subsystem must prefer capabilities from the content repository, but should allow capabilities from the system repository if they can be used to solve an inconsistent class space (uses) issue.

5.6.4 Content + Preferred Provider + System + Local Repository

Option 3 with an additional step. If option 3, step 3 is executed, proceed to step 4 "The Local Repository", even if capabilities were found in the system repository.

5.6.5 All Repositories

Even more radical: Always continue on to the next step even when capabilities are found for all repositories. This is most likely a bad thing to do and has the potential to pull in the world from all registered Repository services unnecessarily during a resolve operation.

Section 5.6.4 is the chosen solution. The Resolver Service Specification makes the following guarantee.

"The Resolver will treat the order of the capabilities as preferences, the first element is more preferred than a later element. The Resolver cannot guarantee that the wiring obeys this preference since there can be other
This is deemed sufficient for the purposes of subsystems. All matching capabilities from the Content, Preferred Provider, System, and Local repositories, and in that order, MUST be provided to the resolver when attempting to resolve a subsystem. An allowable implementation is to return all matching capabilities in the right order within the context of a single call to ResolveContext.findProviders. Furthermore, implementations MAY also return matching capabilities from Repository Services.

6 Considered Alternatives

6.1 Weaving Hooks and DynamicImport-Package

SSU025 – The specification MUST clarify the behaviour in the present of Dynamic-ImportPackage headers defined on content bundles inside scoped Subsystems.

The DynamicImport-Package header will not be addressed by this RFC. For the below reasons, we should wait for a compelling need to arise before attempting to tackle it.

Although it is straightforward to specify well-defined behavior when installing and starting subsystems [1], complexity rapidly ensues when considering what happens when bundles are refreshed, or when children are installed into an already resolved parent subsystem, and how the runtime wiring faithfully represents the deployment manifest [2].

Another concern is that computing the DIP header for applications is not binary compatible [2]. Some sort of opt-in mechanism would need to be devised.

[1] The DIP header would have the same effect as dynamic package imports added by weaving hooks. That is, the imports would only take affect after the subsystem was resolved. This can be accomplished by specifying that the DIP header is only applied to the sharing policy just before entering the STARTING state.

[2] For example, a requirement of a content resource might get wired to an external resource when a local one was preferred.

6.2 Subsystem API to provide more access to information

In addition to the new getDeploymentHeaders() method, defining a SubsystemDTO was considered. It was decided to leave this for the future since nobody was currently asking for it.

### What about provisioning decision information? If we do this, we should tie it into some kind of DTO (dumps the internal info into a DTO).

6.3 Bug 1916 – Define the TCCL for Scoped Subsystems

SSU030 – The specification SHOULD make it possible to define a TCCL policy for proxied interactions between bundle inside a subsystem and for calls out of a subsystem.
### Consider service loader design. This could be done just as a blueprint spec addition. Will depend on whether we went the Virgo route for context bundles. Action: move to Blueprint 1.1?

Bug 1916 summaries the requirement for TCCL support in scoped Subsystems. Essentially, certain legacy libraries, such as Hibernate, depend on using the thread context class loader (TCCL) to load application types. Independently of the mechanism used to define the TCCL, the precise conditions which cause the TCCL to be set on a call out of the scoped application must also be specified. This may depend on whether a service is called or an exported type used directly.

### David suggested there may be some useful info in the Service Loader spec for TCCL.

### Solutions today rely on proxying by component models (e.g. Spring or Blueprint). They also vary in design.

We could standardize something that allows the existing options/approaches. We could try to standardize something lower level that allows more aggressive proxying. Or we could punt, for now...

### 6.4 TCCL Reliance (from Application Domain section)

Libraries not originally designed for use in OSGi can often rely on the use of Thread Context Class Loaders. RFC 133 sought to provide a general solution to the problem and documented the Eclipse Buddy Classloading design. Unfortunately, no worthy solution was identified and so this effort stalled. Approaches based around the concept of isolated applications (precursors to Application Subsystems) have been employed in WebSphere Application Server and Eclipse Virgo. These do not try to be general solutions, but simply seek to provide a TCCL in a deterministic fashion within the context of a scoped application. At a high level, the two approaches are:

1. **Synthetic Context Bundle:** When calling out of a scoped application, Virgo sets the TCCL to be the class loader of a specially created bundle which imports all the packages exported by bundles in the scope in question.

   All types which need to be available to the TCCL must therefore have their packages exported. This places a small restriction on the application: no two bundles in the scope may export the same package.


2. **Calling Bundle:** WebSphere Application Server sets the TCCL to be the class loader of the calling bundle for all calls managed by Blueprint. All types which need to be available to the TCCL must either be defined by the calling bundle or must be exported by the bundle defining the type and imported by the calling bundle.

   This approach aims to give the application developer control over the TCCL without imposing restrictions or overly compromising the application modularity.

   It is important to standardize the TCCL approach or approaches available in Subsystems as the lack of a standard affects portability. For example, an application that relies on the visibility provided by Eclipse Virgo may not have access to all the types it requires when deployed to WebSphere Application Server.

### 6.5 Subsystem Configuration

#### 6.5.1 Requirements

SSU070 – The specification SHOULD make it possible to configure the bundles of a subsystem via configuration admin (e.g. provide a standard form for the bundle locations inside a subsystem).

SSU075 – The specification SHOULD make it possible to provide configuration resources as part of a subsystem archive where those resources are intended to configure bundles of the subsystem, or any bundles within the same region. The will require some means of selecting which configuration admin to use (e.g. one inside the subsystem region).
6.5.2 Problem Description
Configuration of subsystems via configuration admin had to be dropped from Subsystems 1.0 due to time constraints. Time permitting, this RFC should consider two things:

1. How to identify targets of configuration within a subsystem such that configurations provided outside the subsystem can be delivered to the right target within a subsystem.

2. How to provide configurations as resources in a subsystem archive. These configuration should only apply to targets within the subsystem that contributed them or any targets that are in the same containing region.

6.5.3 Technical Solution
The Configuration Admin service is an important aspect of the deployment of an OSGi framework. The Subsystems 1.0 specification did not address how configuration data can be received by bundles which are deployed in a subsystem. The following are the important entities from Configuration Admin that need to be considered when dealing with configuring bundles deployed in a subsystem.

- **Configuration Target** - The target service that will receive the configuration information. For services, there are two types of targets: ManagedServiceFactory or ManagedService objects. Extenders may also define additional configuration targets. For example, Declarative Services specifies a way service components receive configuration data without requiring the registration of ManagedServiceFactory or ManagedService services.
  - Subsystem Configuration Target – The scoped subsystem which contains the Configuration Target.

- **Configuring Bundle** – A bundle that modifies the configuration information through the Configuration Admin service. This bundle is either a management bundle or the bundle for which the configuration information is intended.
  - For subsystems the configuring bundle may be deployed in Subsystem Configuration Target but this is not required. The configuring bundle may also live outside of the Subsystem Configuration Target. If configuration resources are included in a subsystem then the configuring bundle likely will be the subsystems implementation itself.

- **Configuration Extenders** – An Extender, such as Declarative Services, that is responsible for delivering configuration data to the entities they extend, such as service components.
  - For subsystems special consideration is needed allow an extender to deliver configuration data. For example, the Declarative Service specification mandates that Configuration objects must be obtained from the Configuration Admin service using the Bundle Context of the bundle containing the component (i.e. the Configuration Target). This implies that the ConfigurationAdmin service must be available within the context of the Subsystem Configuration Target.

- **Targeted PID** – Specially formatted PIDs that are interpreted by the Configuration Admin service. The target PID scopes the applicability of the PID to a limited set of target bundles.
  - For subsystems a schema is needed to allow configurations to be targeted for specific bundles deployed in scoped subsystems. This important because scoped subsystems provide isolation to a group of bundles. Configurations intended for one Subsystem Configuration Target should not affect targets contained in other Subsystem Configuration Targets.

6.5.3.1 Subsystem Configuration Target
A Configuration Target must not need to be aware of the Subsystem Configuration Target in which they are deployed. Configuration Targets must be able to receive configuration data in the same way regardless of the way they are deployed (in a subsystem or directly into a framework). In order to allow configuring bundles to identify Subsystem Configuration Targets appropriately there needs to be a specified way of identifying which
Subsystem Configuration Target a bundle belongs to. To do this a bundle location schema is defined for bundles that are deployed by the subsystems implementation:

```
subsystem-location '/!' symbolic-name @ 'version'
```

Where subsystem-location is the subsystem location for the Subsystem Configuration Target, symbolic-name is the symbolic of the bundle and version is the version of the bundle. Except for the subsystem region context bundle, all bundles deployed by the subsystems implementation must use this bundle location schema when installing bundles which belong to a subsystem.

### 6.5.4 Delivering Configuration Data

Scoped subsystems provide isolation which allows for a bundle to be deployed multiple times within the same framework instance. This implies that the same Configuration Target may have multiple instances in the same framework instance. In order to deliver configuration data to Subsystem Configuration Targets appropriately a configuring bundle should use fully qualified targeted PIDs. This will ensure that the configuration data only gets delivered to the appropriate Configuration Target.

For example, a configuration target is registered with the PID com.example.web.WebConf by a bundle with the symbolic-name com.acme.example and version 3.2.0 deployed to a scoped subsystem with the location ApplicationX. The following would be the fully qualified target pid to use:

```
com.example.web.WebConf|com.acme.example|3.2.0|ApplicationX
```

XXX - An alternative is to use the concept of Regions from the Configuration Admin service specification. This option requires the use of a security manager to provide isolation. Should we mandate that subsystems implementations grant the ConfigurationPermission["<?subsystem-location>", TARGET] permission to all bundles.

#### 6.5.4.1 Configuration Target Visibility

In order for a Configuration Admin Service implementation to deliver configuration data to configuration targets it must have access to ManagedServiceFactory and ManagedService service registrations contained within a scoped subsystems. In most cases the Configuration Admin implementation will not be contained in the Subsystem Configuration Target and will not have visibility to the necessary services.

XXX – There are two ways I can think of to accomplish this:

1. Have a configuration admin implementation that is subsystems aware. All this means is that is uses the system bundle context in order to discover all MSF and MS services no matter what scope they live in.

2. Have the subsystems implementation somehow grant the configuration admin implementation bundle access to all MSF and MS service registrations.

My preference is to use option 1 since it would not require any strange special holes to be poked through the subsystem sharing policy to expose specific services to specific bundles.

### 6.5.5 Configuration Admin Service Visibility

In some scenarios it is required that the configuration target has access to the Configuration Admin Service. For example, if a bundle manages its own configuration data. An extender is also able to use the Configuration Admin Service database to retrieve and deliver configuration data to the components it is extending. Declarative Services is one such example. The SCR must use the Configuration Target bundle's context in order to retrieve configuration data from the Configuration Admin service. This is to allow for proper security checks and also to allow for accurate matching of targeted PIDs.

In order for these scenarios to work the subsystems implementation must implicitly import the ConfigurationAdmin service into each scoped subsystem. The one exception is if the scoped subsystem is a composite subsystem which exports a Configuration Admin service (XXX – not really sure on this one, perhaps we do not really need to special cases this, would it be bad to export and implicitly import the same service?)
6.6 Configuration Resources

XXX – Left as a brain storming session for the F2F ;-

Need to allow for some configuration resources. Initial though is to have the resources themselves encode the configuration target in their file name:

<bundle-symbolic-name>@<bundle-version>@<PID>.cfg

The cfg file would be a simple properties file where the key is the configuration key and the value is the configuration key value. Perhaps it uses the syntax for capability attributes from the Provide-Capability header except we would have to substitute the ':' type separator with another char (maybe '@'). For example:

ports@List<Long>=1,2,3
server.name=Acme Server

The subsystems implementation would read these cfg resources and populate the configuration admin database with the proper targeted PID configurations. May also need to specify a new resource type of osgi.subsystem.configuration that can have a proper osgi identity and live in a repository and be included in the Subsystem-Content header.

6.7 Manifest matching rules

It's unclear how to match a Deployment.MF to a subsystem in all cases. Although the Symbolic Name and Version are required for the Deployment.MF, they are not required for Subsystem.MF. How do we match them when not specified in the Subsystem.MF? One possibility would be to try to match against defaults and if they do not match, fail the deployment?

### This could come down to a spec clarification. What's the default symbolic name.

7 Security Considerations

The new Subsystem.install(String location, InputStream content, InputStream deploymentManifest) method will inherit the same permission requirements from the other install methods, namely SubsystemPermission[installed subsystem, LIFECYCLE].

The new Subsystem.getDeploymentHeaders() method will require the SubsystemPermission[installed subsystem, METADATA] permission.

8 Javadoc
<table>
<thead>
<tr>
<th>Package Summary</th>
<th>Subsystem Service Package Version 1.1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.service.subsystem</td>
<td>Subsystem Service Package Version 1.1.</td>
</tr>
<tr>
<td>Page</td>
<td>20</td>
</tr>
</tbody>
</table>

Copyright © OSGi Alliance 2013
All Rights Reserved
Package org.osgi.service.subsystem

@org.osgi.annotation.versioning.Version(value="1.1")

Subsystem Service Package Version 1.1.

See: [Description](#)

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subsystem</strong></td>
<td>21</td>
</tr>
</tbody>
</table>

A subsystem is a collection of resources constituting a logical, possibly isolated, unit of functionality.

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SubsystemConstants</strong></td>
<td>37</td>
</tr>
</tbody>
</table>

Defines the constants used by Subsystem service property, manifest header, attribute and directive keys.

<table>
<thead>
<tr>
<th>Enum Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subsystem.State</strong></td>
<td>33</td>
</tr>
</tbody>
</table>

An enumeration of the possible states of a subsystem.

Package org.osgi.service.subsystem Description

Subsystem Service Package Version 1.1.

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.subsystem; version="[1.1,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.subsystem; version="[1.1,1.2)"
```
A subsystem is a collection of resources constituting a logical, possibly isolated, unit of functionality.

A subsystem may be scoped or unscoped. Scoped subsystems are isolated by implicit or explicit sharing policies. Unscoped subsystems are not isolated and, therefore, have no sharing policy. There are three standard types of subsystems.

1. **Application** - An implicitly scoped subsystem. Nothing is exported, and imports are computed based on any unsatisfied content requirements.
2. **Composite** - An explicitly scoped subsystem. The sharing policy is defined by metadata within the subsystem archive.
3. **Feature** - An unscoped subsystem.

Conceptually, a subsystem may be thought of as existing in an isolated region along with zero or more other subsystems. Each region has one and only one scoped subsystem, which dictates the sharing policy. The region may, however, have many unscoped subsystems. It is, therefore, possible to have shared constituents across multiple subsystems within a region. Associated with each region is a bundle whose context may be retrieved from any subsystem within that region. This context may be used to monitor activity occurring within the region.

A subsystem may have children and, unless it's the root subsystem, must have at least one parent. Subsystems become children of the subsystem in which they are installed. Unscoped subsystems have more than one parent if they are installed in more than one subsystem within the same region. The subsystem graph may be thought of as an acyclic digraph with one and only one source vertex, which is the root subsystem. The edges have the child as the head and parent as the tail.

A subsystem has several identifiers.

1. **Location** - An identifier specified by the client as part of installation. It is guaranteed to be unique within the same framework.
2. **ID** - An identifier generated by the implementation as part of installation. It is guaranteed to be unique within the same framework.
3. **Symbolic Name/Version** - The combination of symbolic name and version is guaranteed to be unique within the same region. Although type is not formally part of the identity, two subsystems with the same symbolic names and versions but different types are not considered to be equal.

A subsystem has a well-defined life cycle. Which stage a subsystem is in may be obtained from the subsystem's state and is dependent on which life cycle operation is currently active or was last invoked.

A subsystem archive is a ZIP file having an .esa extension and containing metadata describing the subsystem. The form of the metadata may be a subsystem or deployment manifest, as well as any content resource files. The manifests are optional and will be computed if not present. The subsystem manifest headers may be retrieved in raw or localized forms. There are five standard types of resources that may be included in a subsystem.

1. **Bundle** - A bundle that is not a fragment.
2. **Fragment** - A fragment bundle.
3. **Application Subsystem** - An application subsystem.
4. **Composite Subsystem** - A composite subsystem.
5. **Feature Subsystem** - A feature subsystem.

Resources contained by a subsystem are called constituents. There are several ways a resource may become a constituent of a subsystem:

- A resource is listed as part of the subsystem's content.
- A subsystem resource is a child of the subsystem.
The subsystem has a provision policy of accept dependencies.
A bundle resource is installed using the region bundle context.
A bundle resource is installed using the bundle context of another resource contained by the subsystem.

In addition to invoking one of the install methods, a subsystem instance may be obtained through the service registry. Each installed subsystem has a corresponding service registration. A subsystem service has the following properties.

- **ID** - The ID of the subsystem.
- **Symbolic Name** - The symbolic name of the subsystem.
- **Version** - The version of the subsystem.
- **Type** - The type of the subsystem.
- **State** - The state of the subsystem.

Because a subsystem must be used to install other subsystems, a root subsystem is provided as a starting point. The root subsystem may only be obtained as a service and has the following characteristics.

- The ID is 0.
- The symbolic name is `org.osgi.service.subsystem.root`.
- The version matches this specification's version.
- It has no parents.
- All existing bundles, including the system and subsystem implementation bundles, are constituents.
- The type is `osgi.subsystem.application` with no imports.
- The provision policy is `acceptDependencies`.

### Nested Class Summary

<table>
<thead>
<tr>
<th>Static Enum</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Subsystem.State</code></td>
<td>33</td>
</tr>
</tbody>
</table>

An enumeration of the possible states of a subsystem.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getBundleContext()</code></td>
<td>23</td>
</tr>
<tr>
<td><code>getChildren()</code></td>
<td>23</td>
</tr>
<tr>
<td><code>getConstituents()</code></td>
<td>25</td>
</tr>
<tr>
<td><code>getDeploymentHeaders()</code></td>
<td>25</td>
</tr>
<tr>
<td><code>getLocation()</code></td>
<td>24</td>
</tr>
<tr>
<td><code>getParents()</code></td>
<td>24</td>
</tr>
<tr>
<td><code>getState()</code></td>
<td>25</td>
</tr>
<tr>
<td><code>getSubsystemHeaders(Locale locale)</code></td>
<td>24</td>
</tr>
<tr>
<td><code>getSubsystemId()</code></td>
<td>26</td>
</tr>
</tbody>
</table>
## Method Detail

### getBundleContext

```java
org.osgi.framework.BundleContext getBundleContext()
```

Returns the bundle context of the region within which this subsystem resides.

The bundle context offers the same perspective of any resource contained by a subsystem within the region. It may be used, for example, to monitor events internal to the region as well as external events visible to the region. All subsystems within the same region have the same bundle context. If this subsystem is in a state where the bundle context would be invalid, `null` is returned.

**Returns:**

The bundle context of the region within which this subsystem resides or `null` if this subsystem's state is in `INSTALL_FAILED, UNINSTALLED`.

**Throws:**

- `SecurityException` - If the caller does not have the appropriate `org.osgi.service.subsystem.SubsystemPermission[this,CONTEXT]`, and the runtime supports permissions.

### getChildren

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

Returns the child subsystems of this subsystem.

**Returns:**

The child subsystems of this subsystem. The returned collection is an unmodifiable snapshot of all subsystems that are installed in this subsystem. The collection will be empty if no subsystems are installed in this subsystem.

**Throws:**

- `IllegalStateException` - If this subsystem's state is in `INSTALL_FAILED, UNINSTALLED`.
getSubsystemHeaders

Map<String, String> getSubsystemHeaders(Locale locale)

Returns the headers for this subsystem's subsystem manifest.

Each key in the map is a header name and the value of the key is the corresponding header value. Because header names are case-insensitive, the methods of the map must treat the keys in a case-insensitive manner. If the header name is not found, null is returned. Both original and derived headers will be included in the map.

This method must continue to return the headers while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

Parameters:
locale - The locale for which translations are desired. The header values are translated according to the specified locale. If the specified locale is null or not supported, the raw values are returned. If the translation for a particular header is not found, the raw value is returned.

Returns:
The headers for this subsystem's subsystem manifest. The returned map is unmodifiable.

Throws:
SecurityException - If the caller does not have the appropriate org.osgi.service.subsystem.SubsystemPermission[this,METADATA], and the runtime supports permissions.

getLocation

String getLocation()

Returns the location identifier of this subsystem.

The location identifier is the location that was passed to the install method of the parent subsystem. It is unique within the framework.

This method must continue to return this subsystem's headers while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

Returns:
The location identifier of this subsystem.

Throws:
SecurityException - If the caller does not have the appropriate org.osgi.service.subsystem.SubsystemPermission[this,METADATA], and the runtime supports permissions.

getParents

Collection<Subsystem> getParents()

Returns the parent subsystems of this subsystem.

Returns:
The parent subsystems of this subsystem. The returned collection is an unmodifiable snapshot of all subsystems in which this subsystem is installed. The collection will be empty for the root subsystem; otherwise, it must contain at least one parent. Scoped subsystems always have only one parent. Unscoped subsystems may have multiple parents.
Interface Subsystem

Throws:
   IllegalStateException - If this subsystem's state is in INSTALL_FAILED, UNINSTALLED.

getConstituents

Collection<org.osgi.resource.Resource> getConstituents()

Returns the constituent resources of this subsystem.

Returns:
   The constituent resources of this subsystem. The returned collection is an unmodifiable snapshot of the constituent resources of this subsystem. If this subsystem has no constituents, the collection will be empty.

Throws:
   IllegalStateException - If this subsystem's state is in INSTALL_FAILED, UNINSTALLED.

getDeploymentHeaders

Map<String,String> getDeploymentHeaders()

Returns the headers for this subsystem's deployment manifest.

Each key in the map is a header name and the value of the key is the corresponding header value. Because header names are case-insensitive, the methods of the map must treat the keys in a case-insensitive manner. If the header name is not found, null is returned. Both original and derived headers will be included in the map.

This method must continue to return the headers while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

Returns:
   The headers for this subsystem's deployment manifest. The returned map is unmodifiable.

Throws:
   SecurityException - If the caller does not have the appropriate org.osgi.service.subsystem.SubsystemPermission[this,METADATA], and the runtime supports permissions.
Since: 1.1

getState

Subsystem.State getState()

Returns the current state of this subsystem.

This method must continue to return this subsystem's state while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

Returns:
   The current state of this subsystem.
Interface Subsystem

getSubsystemId

long getSubsystemId()

Returns the identifier of this subsystem.

The identifier is a monotonically increasing, non-negative integer automatically generated at installation time and guaranteed to be unique within the framework. The identifier of the root subsystem is zero.

This method must continue to return this subsystem’s identifier while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

Returns:
The identifier of this subsystem.

getsymbolicName

String getsymbolicName()

Returns the symbolic name of this subsystem.

The subsystem symbolic name conforms to the same grammar rules as the bundle symbolic name and is derived from one of the following, in order.

- The value of the Subsystem-SymbolicName header, if specified.
- The subsystem URI if passed as the location along with the content to the install method.
- Optionally generated in an implementation specific way.

The combination of subsystem symbolic name and version is unique within a region. The symbolic name of the root subsystem is org.osgi.service.subsystem.root.

This method must continue to return this subsystem's symbolic name while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

Returns:
The symbolic name of this subsystem.

getype

String getype()

Returns the type of this subsystem.

This method must continue to return this subsystem's type while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

Returns:
The type of this subsystem.

getVersion

org.osgi.framework.Version getVersion()

Returns the version of this subsystem.
The subsystem version conforms to the same grammar rules as the bundle version and is derived from one of the following, in order.

- The value of the `Subsystem-Version` header, if specified.
- The subsystem URI if passed as the `location` along with the `content` to the `install` method.
- Defaults to `0.0.0`.

The combination of subsystem `symbolic_name` and version is unique within a region. The version of the root subsystem matches this specification's version.

This method must continue to return this subsystem's version while this subsystem is in the `INSTALL_FAILED` or `UNINSTALLED` states.

Returns:

The version of this subsystem.

---

**install**

```java
Subsystem install(String location)
```

Installs a subsystem from the specified location identifier.

This method performs the same function as calling `install(String, InputStream)` with the specified location identifier and `null` as the content.

**Parameters:**

- `location` - The location identifier of the subsystem to install.

**Returns:**

The installed subsystem.

**Throws:**

- `IllegalStateException` - If this subsystem's state is in `INSTALLING`, `INSTALL_FAILED`, `UNINSTALLING`, `UNINSTALLED`.
- `SubsystemException` - If the installation failed.
- `SecurityException` - If the caller does not have the appropriate `org.osgi.service.subsystem.SubsystemPermission[installed subsystem,LIFECYCLE]`, and the runtime supports permissions.

**See Also:**

`install(String, InputStream)`

---

**install**

```java
Subsystem install(String location, InputStream content)
```

Installs a subsystem from the specified content.

The specified location will be used as an identifier of the subsystem. Every installed subsystem is uniquely identified by its location, which is typically in the form of a URI. If the specified location conforms to the `subsystem-uri` grammar, the required symbolic name and optional version information will be used as default values.

If the specified content is `null`, a new input stream must be created from which to read the subsystem by interpreting, in an implementation dependent manner, the specified location.

A subsystem installation must be persistent. That is, an installed subsystem must remain installed across Framework and VM restarts.
All references to changing the state of this subsystem include both changing the state of the subsystem object as well as the state property of the subsystem service registration.

The following steps are required to install a subsystem.

1. If an installed subsystem with the specified location identifier already exists, return the installed subsystem.
2. Read the specified content in order to determine the symbolic name, version, and type of the installing subsystem. If an error occurs while reading the content, an installation failure results.
3. If an installed subsystem with the same symbolic name and version already exists within this subsystem's region, complete the installation with one of the following.
   - If the installing and installed subsystems' types are not equal, an installation failure results.
   - If the installing and installed subsystems' types are equal, and the installed subsystem is already a child of this subsystem, return the installed subsystem.
   - If the installing and installed subsystems' types are equal, and the installed subsystem is not already a child of this subsystem, add the installed subsystem as a child of this subsystem, increment the installed subsystem's reference count by one, and return the installed subsystem.
4. Create a new subsystem based on the specified location and content.
5. If the subsystem is scoped, install and start a new region context bundle.
6. Change the state to INSTALLING and register a new subsystem service.
7. Discover the subsystem's content resources. If any mandatory resource is missing, an installation failure results.
8. Discover the dependencies required by the content resources. If any mandatory dependency is missing, an installation failure results.
9. Using a framework ResolverHook, disable runtime resolution for the resources.
10. For each resource, increment the reference count by one. If the reference count is one, install the resource. If an error occurs while installing a resource, an install failure results with that error as the cause.
11. If the subsystem is scoped, enable the import sharing policy.
12. Enable runtime resolution for the resources.
13. Change the state of the subsystem to INSTALLED.
14. Return the new subsystem.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption. An interrupted thread should result in a org.osgi.service.subsystem.SubsystemException with an InterruptedException as the cause and be treated as an installation failure.

All installation failure flows include the following, in order.

1. Change the state to INSTALL_FAILED.
2. Change the state to UNINSTALLING.
3. All content and dependencies which may have been installed by the installing process must be uninstalled.
4. Change the state to UNINSTALLED.
5. Unregister the subsystem service.
6. If the subsystem is a scoped subsystem then, uninstall the region context bundle.
7. Throw a org.osgi.service.subsystem.SubsystemException with the cause of the installation failure.

Parameters:
location - The location identifier of the subsystem to be installed.
content - The input stream from which this subsystem will be read or null to indicate the input stream must be created from the specified location identifier. The input stream will always be closed when this method completes, even if an exception is thrown.

Returns:
The installed subsystem.

Throws:
IllegalStateException - If this subsystem's state is in INSTALLING, INSTALL_FAILED, UNINSTALLING, UNINSTALLED.
**Interface Subsystem**

SubsystemException - If the installation failed.
SecurityException - If the caller does not have the appropriate org.osgi.service.subsystem.SubsystemPermission[installed subsystem,LIFECYCLE], and the runtime supports permissions.

---

**install**

```java
Subsystem install(String location,
    InputStream content,
    InputStream deploymentManifest)
```

Installs a subsystem from the specified content according to the specified deployment manifest.

This method installs a subsystem using the provided deployment manifest instead of the one in the archive, if any, or a computed one. If the deployment manifest is null, the behavior is exactly the same as in the `install(String, InputStream)` method. Implementations must support deployment manifest input streams in the format described by section 134.2 of the Subsystem Service Specification. If the deployment manifest does not conform to the subsystem manifest (see 134.15.2), the installation fails.

**Parameters:**
- `location` - The location identifier of the subsystem to be installed.
- `content` - The input stream from which this subsystem will be read or null to indicate the input stream must be created from the specified location identifier. The input stream will always be closed when this method completes, even if an exception is thrown.
- `deploymentManifest` - The deployment manifest to use in lieu of the one in the archive, if any, or a computed one.

**Returns:**
The installed subsystem.

**Throws:**
- `IllegalStateException` - If this subsystem's state is in INSTALLING, INSTALL_FAILED, UNINSTALLING, UNINSTALLED.
- `SubsystemException` - If the installation failed.
- `SecurityException` - If the caller does not have the appropriate org.osgi.service.subsystem.SubsystemPermission[installed subsystem,LIFECYCLE], and the runtime supports permissions.

**Since:**
1.1

---

**start**

```java
void start()
```

Starts this subsystem.

The following table shows which actions are associated with each state. An action of Wait means this method will block until a state transition occurs, upon which the new state will be evaluated in order to determine how to proceed. If a state transition does not occur in a reasonable time while waiting then no action is taken and a SubsystemException is thrown to indicate the subsystem was unable to be started. An action of Return means this method returns immediately without taking any other action.

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALLING</td>
<td>Wait</td>
</tr>
<tr>
<td>INSTALLED</td>
<td>Resolve, Start</td>
</tr>
<tr>
<td>INSTALL_FAILED</td>
<td>IllegalStateException</td>
</tr>
<tr>
<td>RESOLVING</td>
<td>Wait</td>
</tr>
<tr>
<td>RESOLVED</td>
<td>Start</td>
</tr>
<tr>
<td>STARTING</td>
<td>Wait</td>
</tr>
</tbody>
</table>
A subsystem must be persistently started. That is, a started subsystem must be restarted across Framework and VM restarts, even if a start failure occurs.

The following steps are required to start this subsystem.

1. Set the subsystem autostart setting to started.
2. If this subsystem is in the RESOLVED state, proceed to step 7.
3. Change the state to RESOLVING.
4. Resolve the content resources. A resolution failure results in a start failure with a state of INSTALLED.
5. Change the state to RESOLVED.
6. If this subsystem is scoped, enable the export sharing policy.
7. Change the state to STARTING.
8. For each eligible resource, increment the active use count by one. If the active use count is one, start the resource. All dependencies must be started before any content resource, and content resources must be started according to the specified start order. If an error occurs while starting a resource, a start failure results with that error as the cause.
9. Change the state to ACTIVE.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption. An interrupted thread should be treated as a start failure with an InterruptedException as the cause.

All start failure flows include the following, in order.

1. If the subsystem state is STARTING then change the state to STOPPING and stop all resources that were started as part of this operation.
2. Change the state to either INSTALLED or RESOLVED.
3. Throw a SubsystemException with the specified cause.

Throws:
- SubsystemException - If this subsystem fails to start.
- IllegalStateException - If this subsystem’s state is in INSTALL_FAILED, UNINSTALLING, or UNINSTALLED, or if the state of at least one of this subsystem’s parents is not in STARTING, ACTIVE.
- SecurityException - If the caller does not have the appropriate org.osgi.service.subsystem.SubsystemPermission [this,EXECUTE], and the runtime supports permissions.

stop

void stop()

Stops this subsystem.

The following table shows which actions are associated with each state. An action of Wait means this method will block until a state transition occurs, upon which the new state will be evaluated in order to determine how to proceed. If a state transition does not occur in a reasonable time while waiting then no action is taken and a SubsystemException is thrown to indicate the subsystem was unable to be stopped. An action of Return means this method returns immediately without taking any other action.
### Interface Subsystem

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALLING</td>
<td>Wait</td>
</tr>
<tr>
<td>INSTALLED</td>
<td>Return</td>
</tr>
<tr>
<td>INSTALL_FAILED</td>
<td>IllegalStateException</td>
</tr>
<tr>
<td>RESOLVING</td>
<td>Wait</td>
</tr>
<tr>
<td>RESOLVED</td>
<td>Return</td>
</tr>
<tr>
<td>STARTING</td>
<td>Wait</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>Stop</td>
</tr>
<tr>
<td>STOPPING</td>
<td>Wait</td>
</tr>
<tr>
<td>UNINSTALLING</td>
<td>IllegalStateException</td>
</tr>
<tr>
<td>UNINSTALLED</td>
<td>IllegalStateException</td>
</tr>
</tbody>
</table>

A subsystem must be persistently stopped. That is, a stopped subsystem must remain stopped across Framework and VM restarts.

All references to changing the state of this subsystem include both changing the state of the subsystem object as well as the state property of the subsystem service registration.

The following steps are required to stop this subsystem.

1. Set the subsystem **autostart setting** to **stopped**.
2. Change the state to **STOPPING**.
3. For each eligible resource, decrement the active use count by one. If the active use count is zero, stop the resource. All content resources must be stopped before any dependencies, and content resources must be stopped in reverse **start order**.
4. Change the state to **RESOLVED**.

With regard to error handling, once this subsystem has transitioned to the **STOPPING** state, every part of each step above must be attempted. Errors subsequent to the first should be logged. Once the stop process has completed, a SubsystemException must be thrown with the initial error as the specified cause.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption, in which case a SubsystemException with an InterruptedException as the cause should be thrown. If an interruption occurs while waiting, this method should terminate immediately. Once the transition to the **STOPPING** state has occurred, however, this method must not terminate due to an interruption until the stop process has completed.

**Throws:**
- SubsystemException - If this subsystem fails to stop cleanly.
- IllegalStateException - If this subsystem's state is in INSTALL_FAILED, UNINSTALLING, or UNINSTALLED.
- SecurityException - If the caller does not have the appropriate org.osgi.service.subsystem.SubsystemPermission[this,EXECUTE], and the runtime supports permissions.

---

#### uninstall

```java
void uninstall()
```

Uninstalls this subsystem.

The following table shows which actions are associated with each state. An action of **Wait** means this method will block until a state transition occurs, upon which the new state will be evaluated in order to determine how to proceed. If a state transition does not occur in a reasonable time while waiting then no action is taken and a SubsystemException is thrown to indicate the subsystem was unable to be uninstalled. An action of **Return** means this method returns immediately without taking any other action.
<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALLING</td>
<td>Wait</td>
</tr>
<tr>
<td>INSTALLED</td>
<td>Uninstall</td>
</tr>
<tr>
<td>INSTALL_FAILED</td>
<td>Wait</td>
</tr>
<tr>
<td>RESOLVING</td>
<td>Wait</td>
</tr>
<tr>
<td>RESOLVED</td>
<td>Uninstall</td>
</tr>
<tr>
<td>STARTING</td>
<td>Wait</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>Stop, Uninstall</td>
</tr>
<tr>
<td>STOPPING</td>
<td>Wait</td>
</tr>
<tr>
<td>UNINSTALLING</td>
<td>Wait</td>
</tr>
<tr>
<td>UNINSTALLED</td>
<td>Return</td>
</tr>
</tbody>
</table>

All references to changing the state of this subsystem include both changing the state of the subsystem object as well as the state property of the subsystem service registration.

The following steps are required to uninstall this subsystem after being stopped if necessary.

1. Change the state to **INSTALLED**.
2. Change the state to **UNINSTALLING**.
3. For each referenced resource, decrement the reference count by one. If the reference count is zero, uninstall the resource. All content resources must be uninstalled before any dependencies.
4. Change the state to **UNINSTALLED**.
5. Unregister the subsystem service.
6. If the subsystem is scoped, uninstall the region context bundle.

With regard to error handling, once this subsystem has transitioned to the **UNINSTALLING** state, every part of each step above must be attempted. Errors subsequent to the first should be logged. Once the uninstall process has completed, a `SubsystemException` must be thrown with the specified cause.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption, in which case a `SubsystemException` with an `InterruptedException` as the cause should be thrown. If an interruption occurs while waiting, this method should terminate immediately. Once the transition to the **UNINSTALLING** state has occurred, however, this method must not terminate due to an interruption until the uninstall process has completed.

Throws:

- `SubsystemException` - If this subsystem fails to uninstall cleanly.
- `SecurityException` - If the caller does not have the appropriate `org.osgi.service.subsystem.SubsystemPermission[null,LIFECYCLE]`, and the runtime supports permissions.
## Enum Subsystem.State

```java
org.osgi.service.subsystem
```

```java
java.lang.Object
  java.lang.Enum<Subsystem.State>
    org.osgi.service.subsystem.Subsystem.State
```

### All Implemented Interfaces:
- Comparable<Subsystem.State>, Serializable

### Enclosing class:
- Subsystem

### Public static enum Subsystem.State

```java
public static enum Subsystem.State extends Enum<Subsystem.State> {
    ...
```

An enumeration of the possible states of a subsystem.

These states are a reflection of what constituent resources are permitted to do and not an aggregation of constituent resource states.

### Enum Constant Summary

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE</td>
<td>The subsystem is now running.</td>
<td>35</td>
</tr>
<tr>
<td>INSTALL_FAILED</td>
<td>The subsystem failed to install.</td>
<td>34</td>
</tr>
<tr>
<td>INSTALLED</td>
<td>The subsystem is installed but not yet resolved.</td>
<td>34</td>
</tr>
<tr>
<td>INSTALLING</td>
<td>The subsystem is in the process of installing.</td>
<td>34</td>
</tr>
<tr>
<td>RESOLVED</td>
<td>The subsystem is resolved and able to be started.</td>
<td>34</td>
</tr>
<tr>
<td>RESOLVING</td>
<td>The subsystem is in the process of resolving.</td>
<td>34</td>
</tr>
<tr>
<td>STARTING</td>
<td>The subsystem is in the process of starting.</td>
<td>35</td>
</tr>
<tr>
<td>STOPPING</td>
<td>The subsystem is in the process of stopping.</td>
<td>35</td>
</tr>
<tr>
<td>UNINSTALLED</td>
<td>The subsystem is uninstalled and may not be used.</td>
<td>35</td>
</tr>
<tr>
<td>UNINSTALLING</td>
<td>The subsystem is in the process of uninstalling.</td>
<td>35</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>36</td>
</tr>
<tr>
<td>static Subsystem.State.values()</td>
<td>35</td>
</tr>
</tbody>
</table>

---

O SGi Javadoc -- 10/30/13
Enum Subsystem.State

Enum Constant Detail

INSTALLING

public static final Subsystem.State INSTALLING

The subsystem is in the process of installing.

A subsystem is in the INSTALLING state when the \texttt{install} method of its parent is active, and attempts are being made to install its content resources. If the install method completes without exception, then the subsystem has successfully installed and must move to the \texttt{INSTALLED} state. Otherwise, the subsystem has failed to install and must move to the \texttt{INSTALL_FAILED} state.

INSTALLED

public static final Subsystem.State INSTALLED

The subsystem is installed but not yet resolved.

A subsystem is in the \texttt{INSTALLED} state when it has been installed in a parent subsystem but is not or cannot be resolved. This state is visible if the dependencies of the subsystem's content resources cannot be resolved.

INSTALL_FAILED

public static final Subsystem.State INSTALL_FAILED

The subsystem failed to install.

A subsystem is in the \texttt{INSTALL_FAILED} state when an unrecoverable error occurred during installation. The subsystem is in an unusable state but references to the subsystem object may still be available and used for introspection.

RESOLVING

public static final Subsystem.State RESOLVING

The subsystem is in the process of resolving.

A subsystem is in the \texttt{RESOLVING} state when the \texttt{start} method is active, and attempts are being made to resolve its content resources. If the resolve method completes without exception, then the subsystem has successfully resolved and must move to the \texttt{RESOLVED} state. Otherwise, the subsystem has failed to resolve and must move to the \texttt{INSTALLED} state.

RESOLVED

public static final Subsystem.State RESOLVED

The subsystem is resolved and able to be started.

A subsystem is in the \texttt{RESOLVED} state when all of its content resources are resolved. Note that the subsystem is not active yet.
Enum Subsystem.State

STARTING

public static final Subsystem.State STARTING

The subsystem is in the process of starting.

A subsystem is in the STARTING state when its start method is active, and attempts are being made to start its content and dependencies. If the start method completes without exception, then the subsystem has successfully started and must move to the ACTIVE state. Otherwise, the subsystem has failed to start and must move to the RESOLVED state.

ACTIVE

public static final Subsystem.State ACTIVE

The subsystem is now running.

A subsystem is in the ACTIVE state when its content and dependencies have been successfully started.

STOPPING

public static final Subsystem.State STOPPING

The subsystem is in the process of stopping.

A subsystem is in the STOPPING state when its stop method is active, and attempts are being made to stop its content and dependencies. When the stop method completes, the subsystem is stopped and must move to the RESOLVED state.

UNINSTALLING

public static final Subsystem.State UNINSTALLING

The subsystem is in the process of uninstalling.

A subsystem is in the UNINSTALLING state when its uninstall method is active, and attempts are being made to uninstall its constituent and dependencies. When the uninstall method completes, the subsystem is uninstalled and must move to the UNINSTALLED state.

UNINSTALLED

public static final Subsystem.State UNINSTALLED

The subsystem is uninstalled and may not be used.

The UNINSTALLED state is only visible after a subsystem's constituent and dependencies are uninstalled. The subsystem is in an unusable state but references to the subsystem object may still be available and used for introspection.

Method Detail

values

public static Subsystem.State[] values()
```java
public static Subsystem.State valueOf(String name)
```
Class SubsystemConstants

org.osgi.service.subsystem

java.lang.Object

- org.osgi.service.subsystem.SubsystemConstants

public class SubsystemConstants
extends Object

Defines the constants used by Subsystem service property, manifest header, attribute and directive keys.

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

Immutable

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static String DEPLOYED_CONTENT</td>
<td>Manifest header identifying the resources to be deployed.</td>
</tr>
<tr>
<td>static String DEPLOYED_VERSION_ATTRIBUTE</td>
<td>Manifest header attribute identifying the deployed version.</td>
</tr>
<tr>
<td>static String DEPLOYMENT_MANIFESTVERSION</td>
<td>Manifest header identifying the deployment manifest version.</td>
</tr>
<tr>
<td>static String PREFERRED_PROVIDER</td>
<td>Manifest header used to express a preference for particular resources to satisfy implicit package dependencies.</td>
</tr>
<tr>
<td>static String PROVISION_POLICY_ACCEPT_DEPENDENCIES</td>
<td>A value for the provision-policy directive indicating the subsystem accepts dependency resources.</td>
</tr>
<tr>
<td>static String PROVISION_POLICY_DIRECTIVE</td>
<td>Manifest header directive identifying the provision policy.</td>
</tr>
<tr>
<td>static String PROVISION_POLICY_REJECT_DEPENDENCIES</td>
<td>A value for the provision-policy directive indicating the subsystem does not accept dependency resources.</td>
</tr>
<tr>
<td>static String PROVISION_RESOURCE</td>
<td>Manifest header identifying the resources to be deployed to satisfy the dependencies of a subsystem.</td>
</tr>
<tr>
<td>static String ROOT_SUBSYSTEM_SYMBOLICNAME</td>
<td>The symbolic name of the root subsystem.</td>
</tr>
<tr>
<td>static String START_ORDER_DIRECTIVE</td>
<td>Manifest header directive identifying the start order of subsystem contents.</td>
</tr>
<tr>
<td>static String SUBSYSTEM_CATEGORY</td>
<td>Manifest header identifying the categories of a subsystem as a comma-delimited list.</td>
</tr>
<tr>
<td>static String SUBSYSTEM_CONTACTADDRESS</td>
<td>Manifest header identifying the contact address where problems with a subsystem may be reported; for example, an email address.</td>
</tr>
<tr>
<td>static String SUBSYSTEM_CONTENT</td>
<td>Manifest header identifying the list of subsystem contents identified by a symbolic name and version.</td>
</tr>
<tr>
<td>static String SUBSYSTEM_COPYRIGHT</td>
<td>Manifest header identifying a subsystem's copyright information.</td>
</tr>
</tbody>
</table>
### Class SubsystemConstants

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SUBSYSTEM_DESCRIPTION</code></td>
<td>Manifest header identifying the human readable description.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_DOCURL</code></td>
<td>Manifest header identifying a subsystem's documentation URL, from which further information about the subsystem may be obtained.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_EXPORTSERVICE</code></td>
<td>Manifest header identifying services offered for export.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_ICON</code></td>
<td>Manifest header identifying the icon URL for the subsystem.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_ID_PROPERTY</code></td>
<td>The name of the service property for the <code>subsystem_id</code>.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_IMPORTSERVICE</code></td>
<td>Manifest header identifying services required for import.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_LICENSE</code></td>
<td>Manifest header identifying a subsystem's license.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_MANIFESTVERSION</code></td>
<td>Manifest header identifying the subsystem manifest version.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_NAME</code></td>
<td>Manifest header identifying the human readable subsystem name.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_STATE_PROPERTY</code></td>
<td>The name of the service property for the subsystem <code>state</code>.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_SYMBOLICNAME</code></td>
<td>Manifest header value identifying the symbolic name for the subsystem.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_SYMBOLICNAME_PROPERTY</code></td>
<td>The name of the service property for the subsystem <code>symbolic_name</code>.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_TYPE</code></td>
<td>Manifest header identifying the subsystem type.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_TYPE_APPLICATION</code></td>
<td>The resource type value identifying an application subsystem.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_TYPE_COMPOSITE</code></td>
<td>The resource type value identifying a composite subsystem.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_TYPE_FEATURE</code></td>
<td>The resource type value identifying an feature subsystem.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_TYPE_PROPERTY</code></td>
<td>The name of the service property for the <code>subsystem_type</code>.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_VENDOR</code></td>
<td>Manifest header identifying a subsystem's vendor.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_VERSION</code></td>
<td>Manifest header value identifying the version of the subsystem.</td>
</tr>
<tr>
<td><code>SUBSYSTEM_VERSION_PROPERTY</code></td>
<td>The name of the service property for the subsystem <code>version</code>.</td>
</tr>
</tbody>
</table>

### Field Detail

**DEPLOYED_CONTENT**

```java
public static final String DEPLOYED_CONTENT = "Deployed-Content"
```

Manifest header identifying the resources to be deployed.
Class SubsystemConstants

DEPLOYED_VERSION_ATTRIBUTE

public static final String DEPLOYED_VERSION_ATTRIBUTE = "deployed-version"

Manifest header attribute identifying the deployed version.

DEPLOYMENT_MANIFESTVERSION

public static final String DEPLOYMENT_MANIFESTVERSION = "Deployment-ManifestVersion"

Manifest header identifying the deployment manifest version. If not present, the default value is 1.

PREFERRED_PROVIDER

public static final String PREFERRED_PROVIDER = "Preferred-Provider"

Manifest header used to express a preference for particular resources to satisfy implicit package dependencies.

PROVISION_POLICY_DIRECTIVE

public static final String PROVISION_POLICY_DIRECTIVE = "provision-policy"

Manifest header directive identifying the provision policy. The default value is rejectDependencies

See Also:
PROVISION_POLICY_ACCEPT_DEPENDENCIES, PROVISION_POLICY_REJECT_DEPENDENCIES

PROVISION_POLICY_ACCEPT_DEPENDENCIES

public static final String PROVISION_POLICY_ACCEPT_DEPENDENCIES = "acceptDependencies"

A value for the provision-policy directive indicating the subsystem accepts dependency resources. The root subsystem has this provision policy.

PROVISION_POLICY_REJECT_DEPENDENCIES

public static final String PROVISION_POLICY_REJECT_DEPENDENCIES = "rejectDependencies"

A value for the provision-policy directive indicating the subsystem does not accept dependency resources. This is the default value.

PROVISION_RESOURCE

public static final String PROVISION_RESOURCE = "Provision-Resource"

Manifest header identifying the resources to be deployed to satisfy the dependencies of a subsystem.
Class SubsystemConstants

START_ORDER_DIRECTIVE

public static final String START_ORDER_DIRECTIVE = "start-order"

Manifest header directive identifying the start order of subsystem contents. There is no default value. Specified values are of type String and must represent an integer.

SUBSYSTEM_CATEGORY

public static final String SUBSYSTEM_CATEGORY = "Subsystem-Category"

Manifest header identifying the categories of a subsystem as a comma-delimited list.

Since:
1.1

SUBSYSTEM_CONTACTADDRESS

public static final String SUBSYSTEM_CONTACTADDRESS = "Subsystem-ContactAddress"

Manifest header identifying the contact address where problems with a subsystem may be reported; for example, an email address.

Since:
1.1

SUBSYSTEM_CONTENT

public static final String SUBSYSTEM_CONTENT = "Subsystem-Content"

Manifest header identifying the list of subsystem contents identified by a symbolic name and version.

SUBSYSTEM_COPYRIGHT

public static final String SUBSYSTEM_COPYRIGHT = "Subsystem-Copyright"

Manifest header identifying a subsystem's copyright information.

Since:
1.1

SUBSYSTEM_DESCRIPTION

public static final String SUBSYSTEM_DESCRIPTION = "Subsystem-Description"

Manifest header identifying the human readable description.

SUBSYSTEM_DOCURL
Manifest header identifying a subsystem's documentation URL, from which further information about the subsystem may be obtained.

Since: 1.1

**SUBSYSTEM_EXPORTSERVICE**

```java
public static final String SUBSYSTEM_EXPORTSERVICE = "Subsystem-ExportService"
```

Manifest header identifying services offered for export.

**SUBSYSTEM_ICON**

```java
public static final String SUBSYSTEM_ICON = "Subsystem-Icon"
```

Manifest header identifying the icon URL for the subsystem.

Since: 1.1

**SUBSYSTEM_ID_PROPERTY**

```java
public static final String SUBSYSTEM_ID_PROPERTY = "subsystem.id"
```

The name of the service property for the `subsystem_id`. The value of this property must be of type `Long`.

**SUBSYSTEM_IMPORTSERVICE**

```java
public static final String SUBSYSTEM_IMPORTSERVICE = "Subsystem-ImportService"
```

Manifest header identifying services required for import.

**SUBSYSTEM_LICENSE**

```java
public static final String SUBSYSTEM_LICENSE = "Subsystem-License"
```

Manifest header identifying a subsystem's license.

Since: 1.1

**SUBSYSTEM_MANIFESTVERSION**

```java
public static final String SUBSYSTEM_MANIFESTVERSION = "Subsystem-ManifestVersion"
```

Manifest header identifying the subsystem manifest version. If not present, the default value is 1.
Class SubsystemConstants

SUBSYSTEM_NAME

public static final String SUBSYSTEM_NAME = "Subsystem-Name"

Manifest header identifying the human readable subsystem name.

SUBSYSTEM_STATE_PROPERTY

public static final String SUBSYSTEM_STATE_PROPERTY = "subsystem.state"

The name of the service property for the subsystem State. The value of this property must be of type Subsystem.State.

SUBSYSTEM_SYMBOLICNAME

public static final String SUBSYSTEM_SYMBOLICNAME = "Subsystem-SymbolicName"

Manifest header value identifying the symbolic name for the subsystem. Must be present.

SUBSYSTEM_SYMBOLICNAME_PROPERTY

public static final String SUBSYSTEM_SYMBOLICNAME_PROPERTY = "subsystem.symbolicName"

The name of the service property for the symbolic name.

ROOT_SUBSYSTEM_SYMBOLICNAME

public static final String ROOT_SUBSYSTEM_SYMBOLICNAME = "org.osgi.service.subsystem.root"

The symbolic name of the root subsystem.

SUBSYSTEM_TYPE

public static final String SUBSYSTEM_TYPE = "Subsystem-Type"

Manifest header identifying the subsystem type.

See Also:

SUBSYSTEM_TYPE_APPLICATION, SUBSYSTEM_TYPE_COMPOSITE, SUBSYSTEM_TYPE_FEATURE

SUBSYSTEM_TYPE_PROPERTY

public static final String SUBSYSTEM_TYPE_PROPERTY = "subsystem.type"

The name of the service property for the subsystem type.

See Also:

SUBSYSTEM_TYPE_APPLICATION, SUBSYSTEM_TYPE_COMPOSITE, SUBSYSTEM_TYPE_FEATURE
Class SubsystemConstants

SUBSYSTEM_TYPE_APPLICATION

public static final String SUBSYSTEM_TYPE_APPLICATION = "osgi.subsystem.application"

The resource type value identifying an application subsystem.

This value is used for the osgi.identity capability attribute type, the SUBSYSTEM_TYPE manifest header
and the SUBSYSTEM_TYPE_PROPERTY service property.

SUBSYSTEM_TYPE_COMPOSITE

public static final String SUBSYSTEM_TYPE_COMPOSITE = "osgi.subsystemcomposite"

The resource type value identifying an composite subsystem.

This value is used for the osgi.identity capability attribute type, the SUBSYSTEM_TYPE manifest header
and the SUBSYSTEM_TYPE_PROPERTY service property.

SUBSYSTEM_TYPE_FEATURE

public static final String SUBSYSTEM_TYPE_FEATURE = "osgi.subsystem.feature"

The resource type value identifying an feature subsystem.

This value is used for the osgi.identity capability attribute type, the SUBSYSTEM_TYPE manifest header
and the SUBSYSTEM_TYPE_PROPERTY service property.

SUBSYSTEM_VENDOR

public static final String SUBSYSTEM_VENDOR = "Subsystem-Vendor"

Manifest header identifying a subsystem's vendor.

Since:

1.1

SUBSYSTEM_VERSION

public static final String SUBSYSTEM_VERSION = "Subsystem-Version"

Manifest header value identifying the version of the subsystem. If not present, the default value is 0.0.0.

SUBSYSTEM_VERSION_PROPERTY

public static final String SUBSYSTEM_VERSIONPROPERTY = "subsystem.version"

The name of the service property for the subsystem version. The value of this property must be of type
Version.
9 Document Support

9.1 References


9.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Graham Charters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM</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>Tom Watson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:tjwatson@us.ibm.com">tjwatson@us.ibm.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>John Ross</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:jwross@us.ibm.com">jwross@us.ibm.com</a></td>
</tr>
</tbody>
</table>

9.3 Acronyms and Abbreviations

ESA – Enterprise Subsystem Archive (or .esa)

9.4 End of Document
Abstract

This document defines the device category for USB devices in OSGi.
0 Document Information

0.1 Table of Contents

0 Document Information .................................................................................................................. 2
  0.1 Table of Contents ........................................................................................................................ 2
  0.2 Terminology and Document Conventions .................................................................................... 3
  0.3 Revision History .......................................................................................................................... 3

1 Introduction .................................................................................................................................. 4

2 Application Domain ..................................................................................................................... 5
  2.1 Terminology + Abbreviations ..................................................................................................... 6

3 Problem Description .................................................................................................................... 7

4 Requirements ............................................................................................................................... 7

5 Technical Solution ....................................................................................................................... 9
  5.1 USBD device Service .................................................................................................................. 9
    5.1.1 Assumptions ........................................................................................................................ 10
    5.1.2 Device Access Category ..................................................................................................... 11
    5.1.3 Service properties from USB Specification .......................................................................... 11
    5.1.4 Other Service properties ....................................................................................................... 14
    5.1.5 Match scale .......................................................................................................................... 15
    5.1.6 Operations .............................................................................................................................. 15
  5.2 USB Serial .................................................................................................................................. 16
    5.2.1 Assumptions ........................................................................................................................ 16
    5.2.2 Optional Device Access Category ......................................................................................... 16
    5.2.3 Optional Service properties .................................................................................................. 16
    5.2.4 Operations .............................................................................................................................. 17
  5.3 Mass Storage ........................................................................................................................... 19
    5.3.1 Assumptions ........................................................................................................................ 19
    5.3.2 Optional Device Access Category ......................................................................................... 19
    5.3.3 Optional Service properties .................................................................................................. 19
    5.3.4 Operations .............................................................................................................................. 20

6 Data Transfer Objects .................................................................................................................. 23

7 Javadoc ......................................................................................................................................... 23

8 Considered Alternatives ............................................................................................................. 31
  8.1 USB.device.interfaceclasses ................................................................................................... 31
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 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>
</table>
| Initial  | April 10, 2013 | Initial version  
Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp |
| v0.2     | July 4, 2013    | – added RFC number to title  
– added 5.1.1.1 Optional Device Access Category  
– modified 5.2.2 Service properties from USB Specification  
Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp |
| v0.3     | Sept. 9, 2013   | – modified based on the F2F meeting in Paris  
Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp |
| v0.4     | Nov. 19, 2013   | – modified based on the F2F meeting in Hursley  
Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp |
| v0.5     | Nov. 19, 2013   | – Updated Javadoc section  
Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp |
## 1 Introduction

OSGi Device Access Specification defines a unified and sophisticated way to handle devices attached to a residential gateway or devices found in the home network by using various protocols such as USB, Zigbee, ZWave, KNX, UPnP etc. However, OSGi Device Access Specification clearly declare that Device Category must be defined outside of OSGi Device Access Specification.

Recently, OSGi is gaining popularity as enabling technology for building embedded system in residential
market. It gets popular that a HGW has USB interfaces and the needs of handling USB devices attached to a residential gateway is increased.

This RFC defines a device category for USB devices.

## 2 Application Domain

Currently there are several standardization bodies such as OSGiA, HGI, BBF, which deal with the deployment of services in an infrastructure based on the usage of a Residential Gateway running OSGi as Execution Platform.

In order to realize services which access not only IP devices but also non-IP devices connected to the residential gateway, there are several protocols for home networks, such as ZigBee, Z-Wave, KNX/EHS, ECHONET, ECHONET-LITE, etc.. While some residential gateways support those protocols on themselves, others do not. Many residential gateways have USB interfaces and there exist USB dongles which support those protocols. Therefore, there is a need to support those protocols using USB dongles attached to a residential gateway (Fig. 1). In addition, most of USB dongles can be controlled through Serial Communication.

The existing OSGi specifications which address related topics are:

- Device Access Specification focuses on the dynamic discovery of the proper driver when a new device it attached/connected to the residential gateway
2.1 Terminology + Abbreviations

- Base Drivers: see \textsuperscript{\textit{103.4.2.1}} in OSGi Device Access Specification \[3\]
- Refining Drivers: see \textsuperscript{\textit{103.4.2.2}} in OSGi Device Access Specification \[3\]
- Match value: the value match() method of a Driver service registered by the refining driver bundle returns. Matching is explained in \textsuperscript{\textit{103.7.2 The Device Attachment Algorithm}} in OSGi Device Access Specification \[3\]
- Device Descriptor: see \textsuperscript{\textit{9.6.1}} in Universal Serial Bus Specification[4]

\textit{Fig 1 USB Dongles and Residential gateway}
3 Problem Description

The existing OSGi Device Access Specification provides the unified way to installation and activation of driver bundles. However, the OSGi Device Access Specification declares the device category for specific devices must be defined outside of itself. Currently, no device category for USB devices has been defined yet.

The lack of the device category for USB devices causes the following problems.

[Problem 1] The developer of a refining driver bundle, which registers a Driver service at its activation, cannot design and implement Driver#attach(ServiceReference) method without knowledge of service properties set to the Device service registered by a USB base driver.

[Problem 2] The developer of a refining driver bundle, which registers a Driver service at its activation, cannot design and implement Driver#match(ServiceReference) method without knowledge of service properties set to the Device service registered by a USB base driver and without the definition of match values to be returned.

In other words, without the device category for USB devices, a refining driver bundle developed by developer A can cooperate with the USB base driver bundle developed by the same developer A but cannot cooperate with the USB base driver bundles developed by the different developer B.

4 Requirements

[REQ_1] The solution MUST be compatible with OSGi Device Access Specification.

[REQ_2] The solution MUST define the details of the registration of a Device service by a USB base driver bundle when a USB device is attached.

[REQ_2-1] The solution MUST define the service interface under which the Device service is registered.

[REQ_2-2] The solution MUST define the service properties with which the Device service is registered: A set of service properties, their data types, and semantics, each of which must be declared as either MANDATORY or OPTIONAL.

[REQ_3] The solution MUST define the way how a driver bundle controls an attached USB device which can be controlled through Serial communication.

[REQ_4] The solution MAY define a range of match values specific to this device category.
[REQ_5] The range of match values MUST be sufficient to describe the required range of native serial drivers specified by the HGI, especially the following ones:

- Class drivers for Human Interface Device (HID) and Communications Device Class (CDC) \(^1\)
- Drivers for FTDI Virtual Com Ports with a variable list of supported USB Vendor Identifiers and Product Identifiers\(^2\).
- Drivers for Silicon Labs CP210x USB to UART bridge and CP2110 HID USB to UART bridge\(^3\).
- USB drivers for Prolific PL-2303 USB to Serial Bridge Controller\(^4\).

\(^1\) [http://www.usb.org/developers/devclass_docs](http://www.usb.org/developers/devclass_docs) approved for details of USB device classes

\(^2\) [http://www.ftdichip.com/Drivers/VCP.htm](http://www.ftdichip.com/Drivers/VCP.htm)


5 Technical Solution

USB device category defines the following elements:

1. An interface that all devices belonging to this category must implement.

2. A set of service registration properties, their data types, and semantics, each of which must be declared as either MANDATORY or OPTIONAL for this device category.

3. A range of match values specific to this device category.

This document defines other elements for some specific USB classes, because of there are clear use cases (Fig 2). This document does not define the details of all USB classes, they are roles of refining drivers. Otherwise, future specification about USB may provide those definitions that define that.

5.1 USBDevice Service

The device services are registered in the OSGi service registry with org.osgi.service.usb.USBDriver interface. The service is registered by a USB base driver bundle when a USB device is attached. A USB base driver bundle must implement
org.osgi.service.usb.USBD"vice interface and register the OSGi service under org.osgi.service.usb.USBD"vice. Refining drivers can find USB devices via USBDevice services and identify the device. The USBDevice service has a set of properties (Fig 3).

Universal Serial Bus Specification (USB Specification) [4] defines that a USB device has USB interface(s). A USB base driver bundle must register USBDevice services number of USB interfaces. A USBDevice service has information that contains a USB device information and a USB interface information.

5.1.1 Assumptions

The USB base driver may need native drivers such as kernel drivers on Linux (Fig 4). This document has a precondition that there are native drivers. It is out of scope how to install native drivers.
5.1.2 Device Access Category

The device access category is called "USB". The category name is defined as a value of USBDevice.DEVICE_CATEGORY constant. It can be used as a part of org.osgi.service.device.Constants.DEVICE_CATEGORY service key value. The category imposes these following specification rules.

- USBDevice.DEVICE_CATEGORY — MANDATORY property. The value is "USB". Constant for the value of the service property DEVICECATEGORY used for all USB devices. A USB base driver bundle must set this property key.

5.1.3 Service properties from USB Specification

Universal Serial Bus Specification (The USB Specification)[4] defines a device descriptor. USB devices report their attributes using descriptors. USBDevice service has some properties from the USB device descriptor. Table 1 shows them.

Table 1: Device Descriptor and Service Property

<table>
<thead>
<tr>
<th>Device Descriptor's Field from USB Spec.</th>
<th>USB property</th>
<th>Device Category's service property</th>
<th>M/O</th>
<th>Java type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLength</td>
<td>none</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>bDescriptorType</td>
<td>none</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>bcdUSB</td>
<td>USB.device.bcdUSB</td>
<td></td>
<td>MO</td>
<td>Integer</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Format</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>bDeviceClass</code></td>
<td><code>USB.device.bDeviceClass</code></td>
<td><code>M</code> Integer</td>
<td><code>String</code></td>
<td></td>
</tr>
<tr>
<td><code>bDeviceSubClass</code></td>
<td><code>USB.device.bDeviceSubClass</code></td>
<td><code>M</code> Integer</td>
<td><code>String</code></td>
<td></td>
</tr>
<tr>
<td><code>bDeviceProtocol</code></td>
<td><code>USB.device.bDeviceProtocol</code></td>
<td><code>M</code> Integer</td>
<td><code>String</code></td>
<td></td>
</tr>
<tr>
<td><code>bMaxPacketSize0</code></td>
<td><code>noneUSB.device.bMaxPacketSize0</code></td>
<td><code>-O</code>  Integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>idVendor</code></td>
<td><code>USB.device.idVendor</code></td>
<td><code>M</code> Integer</td>
<td><code>String</code></td>
<td></td>
</tr>
<tr>
<td><code>idProduct</code></td>
<td><code>USB.device.idProduct</code></td>
<td><code>M</code> Integer</td>
<td><code>String</code></td>
<td></td>
</tr>
<tr>
<td><code>bcdDevice</code></td>
<td><code>USB.device.bcdDevice</code></td>
<td><code>M</code> Integer</td>
<td><code>String</code></td>
<td></td>
</tr>
<tr>
<td><code>iManufacturer</code></td>
<td><code>USB.device.iManufacturer</code></td>
<td><code>O</code> String</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>iProduct</code></td>
<td><code>USB.device.iProduct</code></td>
<td><code>O</code> String</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>iSerialNumber</code></td>
<td><code>USB.device.iSerialNumber</code></td>
<td><code>O</code> String</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>bNumConfigurations</code></td>
<td><code>noneUSB.device.bNumConfigurations</code></td>
<td><code>-O</code>  Integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- `USB.device.bcdUSB` — **MANDATORY** property key. The value is `IntegerString`, the 4-digit BCD format.
  - Example: `0x0210`
- `USB.device.bDeviceClass` — **MANDATORY** property key. The value is `IntegerString`, hexadecimal, 2-digits.
  - Example: `0xff`
- `USB.device.bDeviceSubClass` — **MANDATORY** property key. The value is `IntegerString`, hexadecimal, 2-digits.
  - Example: `0xff`
- `USB.device.bDeviceProtocol` — **MANDATORY** property key. The value is `IntegerString`, hexadecimal, 2-digits.
  - Example: `0xff`
• **USB.device.bMaxPacketSize0** – **OPTIONAL** property key. The value is Integer.

• **USB.device.idVendor** – **MANDATORY** property key. The value is Integer, hexadecimal, 4-digits.
  - Example: "0x0403"

• **USB.device.idProduct** – **MANDATORY** property key. The value is Integer, hexadecimal, 4-digits.
  - Example: "0x8372"

• **USB.device.bcdDevice** – **MANDATORY** property key. The value is Integer, the 4-digit BCD format.
  - Example: 0x0200

• **USB.device.iManufacturer** – **OPTIONAL** property key. The value is String of indicated in iManufacturer. (The value is not the index.)
  - Example: "Buffalo Inc."

• **USB.device.iProduct** – **OPTIONAL** property key. The value is String of indicated in iProduct. (The value is not the index.)
  - Example: "USB2.0 PC Camera"

• **USB.device.iSerialNumber** – **OPTIONAL** property key. The value is String of indicated in iSerialNumber. (The value is not the index.)
  - Example: "57B0002600000001"

• **USB.device.bNumConfigurations** – **OPTIONAL** property key. The value is Integer.

According to the USB Specification, a device descriptor has some interface descriptors. Refining drivers need each interface descriptors’ bInterfaceClass, bInterfaceSubClass and bInterfaceProtocol to identify devices. So these fields add to the service properties (see Table 2).

---

**Table 2: Interface Descriptor and Service Property**

<table>
<thead>
<tr>
<th>Interface Descriptor's Field from USB Spec.</th>
<th>USB Device Category's service property</th>
<th>M/O</th>
<th>Java type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLength</td>
<td>none</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>bDescriptorType</td>
<td>none</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>bInterfaceNumber</td>
<td><strong>USB.device.bInterfaceNumber</strong></td>
<td>-M</td>
<td>-Integer</td>
</tr>
</tbody>
</table>
### 5.1.4 Other Service properties

Some other service properties are needed to identify and access a device by refining drivers.

<table>
<thead>
<tr>
<th>Service Properties</th>
<th>Default</th>
<th>Value</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bAlternateSetting</strong></td>
<td>none</td>
<td>-O</td>
<td>Integer</td>
</tr>
<tr>
<td><strong>bNumEndpoints</strong></td>
<td>none</td>
<td>-O</td>
<td>Integer</td>
</tr>
<tr>
<td><strong>bInterfaceClass</strong></td>
<td>USB.device.bInterfaceClass</td>
<td>M</td>
<td>Integer[],  String</td>
</tr>
<tr>
<td><strong>bInterfaceSubClass</strong></td>
<td>USB.device.bInterfaceSubClass</td>
<td>M</td>
<td>String</td>
</tr>
<tr>
<td><strong>bInterfaceProtocol</strong></td>
<td>USB.device.bInterfaceProtocol</td>
<td>M</td>
<td>String</td>
</tr>
<tr>
<td><strong>iInterface</strong></td>
<td>none</td>
<td>-O</td>
<td>String</td>
</tr>
</tbody>
</table>

- **USB.device.bInterfaceNumber** - MANDATORY property key. The value is Integer.
- **USB.device.bAlternateSetting** - OPTIONAL property key. The value is Integer.
- **USB.device.bNumEndpoints** - OPTIONAL property key. The value is Integer.
- **USB.device.bInterfaceClass** - MANDATORY property key. The value is String, hexadecimal, 2-digits.
  - Example: “ff”
- **USB.device.bInterfaceSubClass** - MANDATORY property key. The value is String, hexadecimal, 2-digits.
  - Example: “ff”
- **USB.device.bInterfaceProtocol** - MANDATORY property key. The value is String, hexadecimal, 2-digits.
  - Example: “ff”
- **USB.device.iInterface** - OPTIONAL property key. The value is String of indicated in iInterface. (The value is not the index.) USB interface and is combined the interface’s bInterfaceClass (2-digits), bInterfaceSubClass (2-digits) and bInterfaceProtocol (2-digits).
  - Example: {0x080000, 0x0a00ff} each MANDATORY property key. The property value is Integer[], hexadecimal, 6-digits. Each Integer responds to USB.device.interfaceclasses.
<table>
<thead>
<tr>
<th>Service property</th>
<th>M/O</th>
<th>Java type</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB.device.bus</td>
<td>M</td>
<td>Integer</td>
</tr>
<tr>
<td>USB.device.address</td>
<td>M</td>
<td>Integer</td>
</tr>
</tbody>
</table>

- **USB.device.bus** — MANDATORY property key. The value is Integer. Used to identify USB devices with same VID / PID. The value is the ID of the USB bus assigned when connecting the USB device. USB bus ID is integer (001-127). The USB bus ID does not change while the USB device remains connected.
  - Example: 3

- **USB.device.address** — MANDATORY property key. The value is Integer. Used to identify USB devices with same VID / PID. The value is the ID of the USB address assigned when connecting the USB device. USB address is integer (001-127). The USB address does not change while the USB device remains connected.
  - Example: 2

### 5.1.5 Match scale

When the driver service is registered by the driver bundle, the Device Manager calls Driver#match() with the argument of the USBDevice service's ServiceReference. The driver responds with the value based on below scale.

- **MATCH_MODEL** — Constant for the USB device match scale, indicating a match with USB.device.idVendor and USB.device.idProduct. Value is 10.

- **MATCH_PROTOCOL** — Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass, USB.device.bDeviceSubClass and USB.device.bDeviceProtocol, or a match with bInterfaceClass, bInterfaceSubClass and bInterfaceProtocol in one of USB.device.interfaceclasses. Value is 7.

- **MATCH_SUBCLASS** — Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass and USB.device.bDeviceSubClass, or a match with bInterfaceClass and bInterfaceSubClass in one of USB.device.interfaceclasses. Value is 5.

- **MATCH_CLASS** — Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass, or a match with bInterfaceClass in one of USB.device.interfaceclasses. Value is 3.

### 5.1.6 Operations

Figure 5 describes a mechanism to handle USB devices. When a USB device is attached, a USB base driver bundle recognizes it via native device drivers and gets information from the USB device. The USB base driver bundle registers a USBDevice service with service properties from the information gained.
5.2 USB Serial

This section is defined for USB devices have a serial communication function.

5.2.1 Assumptions

Many residential gateways have USB interfaces. And USB dongles support several protocols for home networks, such as ZigBee, Z-Wave, KNX/EHS, ECHONET, ECHONET-LITE, etc. Most of USB dongles can be controlled through Serial Communication. When a USB dongle is connected, the device is mapped to a COM port automatically by native libraries in OS. This mechanism is out of scope, those libraries are preconditions.

5.2.2 Optional Device Access Category

In this section, an optional device access category is defined.

- **USBDevice.DEVICE_CATEGORY_SERIAL** — OPTIONAL property. The value is "Serial". Constant for the value of the service property DEVICE_CATEGORY used for a USB device which has a serial communication function such as a USB dongle. Such a USB base driver bundle must set this property key and serial.comport property. This device category's value may be used independently of USB. This value is defined because of some USB devices have a serial communication function.

5.2.3 Optional Service properties

Some other service properties are needed to identify and access a device by refining drivers.
Table 4: Optional service properties

<table>
<thead>
<tr>
<th>Service property</th>
<th>M/O</th>
<th>Java type</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial.comport</td>
<td>O</td>
<td>String</td>
</tr>
</tbody>
</table>

- serial.comport  __OPTIONAL__ Property key. The value is String. The USB Device has a serial communication function, set the value that represents the COM port. If the USB device does not have a serial communication function, this key and value is not set. The driver can communicate through Java Communications API with this value. Set this value "portName" of javax.comm.CommPortIdentifier.getPortIdentifier(String portName). Then serial communication is possible. Serial.comport value's format must be equal to the "portName" format. If a USB base driver sets this property, USBDevice.DEVICE_CATEGORY_SERIAL must be set to DEVICE_CATEGORY.

  - Example1: "/dev/ttyUSB0"
  - Example2: "COM5"
  - Example3: "/dev/tty.usbserial-XXXXXX"

5.2.4 Operations

5.2.4.1 USB serial example 1

Figure 6 describes a sample operation to handle USB serial devices.

0. A USB base driver is tracking OS events. Native device driver such as kernel modules in Linux can detect and communicate with USB devices and allocate a USB device or a USB interface to a device file (COM port).

1. USB dongle 1 is connected, native device drivers allocate the device to /dev/ttyUSB0, the USB base driver catches the event and gets information about the device, then the USB base driver registers a USBDevice service with service properties that have DEVICE_CATEGORY ("USB", "Serial"): serial.comport, bDeviceClass, idVendor and so on. NOTE; the USB base driver must handle file.separator properly.

2. USB dongle 1 is removed, native device drivers remove it, the USB base driver unregister the USBDevice service.

3. USB dongle 1 (same device) is connected again, native device driver allocate it to /dev/ttyUSB2 (not same path), the USB base driver should get information about that device again and registers USBDevice service. NOTE; It is not guaranteed that the same device is the same COM port.
5.2.4.2 USB serial example 2

Figure 7 describes another sample operation to handle USB Serial devices.

1. Same as example 1.

2. Same as example 1.

3. USB dongle 2 (not same device) is connected, native device driver allocate it to /dev/ttyUSB0 (same path), the USB base driver registers USBDevice service. NOTE: It is not guaranteed that same COM port is same device.
5.3 Mass Storage

This section is defined for USB devices that are Mass Storage.

5.3.1 Assumptions

When a USB storage is connected, the directory is mounted automatically or manually via some native libraries in OS. This mechanism is out of scope, those libraries are preconditions.

5.3.2 Optional Device Access Category

In this section, an optional device access category is defined.

- USBDevice.DEVICECATEGORY MASSSTORAGE = OPTIONAL property. The value is "MassStorage". Constant for the value of the service property DEVICECATEGORY used for a USB device which is a MassStorage Class in USB Specification[4] such as a USB storage. Such a USB base driver bundle must set this property key and set massstorage.mountpoints property while the device is mounted.

5.3.3 Optional Service properties

Some other service properties are needed to identify and access a device by refining drivers.

Table 5: Optional service properties

<table>
<thead>
<tr>
<th>Service property</th>
<th>M/O</th>
<th>Java type</th>
</tr>
</thead>
</table>

Fig 7: USB Serial Example 2
massstorage.mountpoints: String[]

- **massstorage.mountpoints** is an OPTIONAL property key. The value is String[]. If the USB device is Mass Storage Class, set the value that represents the mount point (a path to the USB storage) in OS. If the USB device is not Mass Storage Class, this key and value is not set. The driver can read and write the USB storage through standard File API with this value. Set this value "pathname" of java.io.File(String pathname). Then file access is possible. Massstorage.mountpoints's format must be equal to the "pathname" format. If a USB base driver sets this property, USBDevice.DEVICE_CATEGORY_MASSSTORAGE must be set to DEVICE_CATEGORY.

  - Example 1: \
    ```
    {"/mnt/media/usb-storage-01/"}
    ```

  - Example 2: \
    ```
    {"D:\Java"}
    ```

### 5.3.4 Operations

#### 5.3.4.1 USB storage example 1

Figure 8 describes a sample operation to handle USB storage devices.

1. A USB base driver is tracking OS events. There are native device drivers and native libraries in OS. Native device drivers such as kernel modules in Linux can detect and communicate with USB devices or USB and allocate it to a device file. Native libraries such as autofs in Linux auto-mount certain virtual devices to mount points.

2. USB storage 1 is connected, native device drivers allocate it to /dev/sda and native libraries auto-mount the virtual device to /mnt/usb0, the USB base driver catches the mount event and gets information about the device, then the USB base driver registers a USBDevice service with service properties that have DEVICE_CATEGORY ({"USB", "MassStorage"}), massstorage.mountpoints, bDeviceClass, idVendor and so on. NOTE; the USB base driver must handle file.separator properly.

3. A user unmount /mnt/usb0 manually, the USB base driver modifies the service property, massstorage.mountpoints is removed.

4. USB storage 1 is removed, native libraries remove the virtual device, the USB base driver unregister the USBDevice service.

5. USB storage 1 (same device) is connected again, native device drivers allocate it to /dev/sdb (not same path) and native libraries mount it to /mnt/usb3 (not same path), the USB base driver should get information about that device again and registers USBDevice service. NOTE; It is not guaranteed that same device is same mountpoint.

6. A user re-mounts /mnt/usb3 to /mnt/myusb, the USB base driver modifies the service property, massstorage.mountpoints is changed from {"/mnt/usb3"} to {"/mnt/myusb"}. 

Copyright © NTT Corporation 2014 All Rights Reserved
5.3.4.2 USB storage example 2

Figure 9 describes another sample operation to handle USB storage devices.

1. USB storage 1 is-connected that has 3 partitions is connected, native device drivers allocate each partition to /dev/sda1, /dev/sda2 and /dev/sda3 and native libraries auto-mount each virtual device to /mnt/usb0, none (not mounted) and /mnt/myusb, the USB base driver catches the mount event and gets information about the device, then the USB base driver registers a USBDevice service with service properties that have DEVICECATEGORY, massstorage.mountpoints ({"/mnt/usb0"}, bDeviceClass, idVendor and so on. NOTE; the USB base driver must handle file.separator properly.

2. A user unmount /mnt/myusb (/dev/sda3) manually, the USB base driver modifies the service property, massstorage.mountpoints is changed to {"/mnt/usb0"}.

3. A user mount /dev/sda2 (not same partition) to /mnt/myusb (same mountpoint), the USB base driver modifies the service property, massstorage.mountpoints is changed to {"/mnt/usb0", "/mnt/myusb"}. NOTE; It is not guaranteed that same mountpoint is same device. If USB Storage has multiple partitions, massstorage.mountpoints does not indicates which partition represents which partition.

---

Copyright © NTT Corporation 2014       All Rights Reserved
5.3.4.3 USB storage example 3

Figure 10 describes another sample operation to handle USB storage devices.

0. A USB base driver is tracking OS events. Native device drivers such as kernel modules in Linux can detect and communicate with USB devices and allocate it to a device file. Native libraries such as autofs in Linux auto-mount certain virtual devices to mount points. In this example, there are native device drivers in OS and native libraries are not installed.

1. USB storage 1 is connected, native device drivers allocate it to /dev/sda, the USB base driver catches the event and gets information about the device, then the USB base driver registers a USBDevice service with service properties that have DEVICECATEGORY ("USB", "MassStorage"), bDeviceClass, idVendor and so on. NOTE; the USB base driver must handle file.separator properly.

2. A user installs and starts the native libraries.

3. Thos libraries mount the virtual device to /mnt/usb0, the USB base driver catches the mount event and modifies the service property, massstorage.mountpoints ("/mnt/usb0") is added.
6 Data Transfer Objects

This RFC will not provide Data Transfer Objects.

7 Javadoc
## Package Summary

<table>
<thead>
<tr>
<th>Package</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.service.usb</td>
<td>25</td>
</tr>
</tbody>
</table>
## Package org.osgi.service.usb

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>USBDevice</strong></td>
<td>26</td>
</tr>
</tbody>
</table>

Represented a USB device.
public interface USBDevice

Represents a USB device. For each USB device, an object is registered with the framework under the USBDevice interface. A USB base driver must implement this interface. The values of the USB property names are defined by the USB Implementers Forum, Inc. The package name is org.osgi.service.usb.

### Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String ALTERNATE_SETTING</td>
<td>String</td>
<td>OPTIONAL property key.</td>
<td>29</td>
</tr>
<tr>
<td>String COM_PORT</td>
<td>String</td>
<td>OPTIONAL Property key.</td>
<td>31</td>
</tr>
<tr>
<td>String DEVICE_CATEGORY</td>
<td>String</td>
<td>MANDATORY property.</td>
<td>27</td>
</tr>
<tr>
<td>String DEVICE_CATEGORY_MASSSTORAGE</td>
<td>String</td>
<td>OPTIONAL Property.</td>
<td>31</td>
</tr>
<tr>
<td>String DEVICE_CATEGORY_SERIAL</td>
<td>String</td>
<td>OPTIONAL Property.</td>
<td>31</td>
</tr>
<tr>
<td>String DEVICE_CLASS</td>
<td>String</td>
<td>MANDATORY property key.</td>
<td>27</td>
</tr>
<tr>
<td>String DEVICE_MAXPACETSIZETO</td>
<td>String</td>
<td>OPTIONAL property key.</td>
<td>28</td>
</tr>
<tr>
<td>String DEVICE_PROTOCOL</td>
<td>String</td>
<td>MANDATORY property key.</td>
<td>28</td>
</tr>
<tr>
<td>String DEVICE_SUBCLASS</td>
<td>String</td>
<td>MANDATORY property key.</td>
<td>28</td>
</tr>
<tr>
<td>String INTERFACE_CLASS</td>
<td>String</td>
<td>MANDATORY property key.</td>
<td>29</td>
</tr>
<tr>
<td>String INTERFACE_DESCRIPTION</td>
<td>String</td>
<td>OPTIONAL property key.</td>
<td>30</td>
</tr>
<tr>
<td>String INTERFACE_NUMBER</td>
<td>String</td>
<td>MANDATORY property key.</td>
<td>29</td>
</tr>
<tr>
<td>String INTERFACE_PROTOCOL</td>
<td>String</td>
<td>MANDATORY property key.</td>
<td>30</td>
</tr>
<tr>
<td>String INTERFACE_SUBCLASS</td>
<td>String</td>
<td>MANDATORY property key.</td>
<td>30</td>
</tr>
<tr>
<td>String MANUFACTURER</td>
<td>String</td>
<td>OPTIONAL property key.</td>
<td>28</td>
</tr>
<tr>
<td>int MATCH_CLASS</td>
<td>int</td>
<td>Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass, or a match with bInterfaceClass in one of USB.device.interfaceclasses.</td>
<td>31</td>
</tr>
<tr>
<td>int MATCH_MODEL</td>
<td>int</td>
<td>Constant for the USB device match scale, indicating a match with USB.device.idVendor and USB.device.idProduct.</td>
<td>30</td>
</tr>
</tbody>
</table>
### Field Detail

#### DEVICE_CATEGORY

```java
public static final String DEVICECATEGORY = "USB"
```

MANDATORY property. The value is "USB". Constant for the value of the service property DEVICE_CATEGORY used for all USB devices. A USB base driver bundle must set this property key. See Also org.osgi.service.device.Constants.DEVICE_CATEGORY

#### USB_RELEASE_NUMBER

```java
public static final String USBRELEASENUMBER = "USB.device.bcdUSB"
```

OPTIONAL property key. Value is "USB.device.bcdUSB". The value is String, the 4-digit BCD format. Example: "0210"

#### DEVICE_CLASS

```java
public static final String DEVICECLASS = "USB.device.bDeviceClass"
```
Interface USBDevice

MANDATORY property key. Value is "USB.device.bDeviceClass". The value is String, hexadecimal, 2-digits. Example: "ff"

DEVICE_SUBCLASS

public static final String DEVICE_SUBCLASS = "USB.device.bDeviceSubClass"

MANDATORY property key. Value is "USB.device.bDeviceSubClass". The value is String, hexadecimal, 2-digits. Example: "ff"

DEVICE_PROTOCOL

public static final String DEVICE_PROTOCOL = "USB.device.bDeviceProtocol"

MANDATORY property key. Value is "USB.device.bDeviceProtocol". The value is String, hexadecimal, 2-digits. Example: "ff"

DEVICE_MAXPACETSIZE0

public static final String DEVICE_MAXPACETSIZE0 = "USB.device.bMaxPacketSize0"

OPTIONAL property key. Value is "USB.device.bMaxPacketSize0". The value is Integer.

VID

public static final String VID = "USB.device.idVendor"

MANDATORY property key. Value is "USB.device.idVendor". The value is String, hexadecimal, 4-digits. Example: "0403"

PID

public static final String PID = "USB.device.idProduct"

MANDATORY property key. Value is "USB.device.idProduct". The value is String, hexadecimal, 4-digits. Example: "8372"

RELEASE_NUMBER

public static final String RELEASE_NUMBER = "USB.device.bcdDevice"

MANDATORY property key. Value is "USB.device.bcdDevice". The value is String, the 4-digit BCD format. Example: "0200"

MANUFACTURER

public static final String MANUFACTURER = "USB.device.iManufacturer"
OPTIONAL property key. Value is "iManufacturer". The value is String of indicated in iManufacturer. (The value is not the index.) Example: "Buffalo Inc."

PRODUCT

public static final String PRODUCT = "USB.device.iProduct"

OPTIONAL property key. Value is "iProduct". The value is String of indicated in iProduct. (The value is not the index.) Example: "USB2.0 PC Camera"

SERIALNUMBER

public static final String SERIALNUMBER = "USB.device.iSerialNumber"

OPTIONAL property key. Value is "USB.device.iSerialNumber". The value is String of indicated in iSerialNumber. (The value is not the index.) Example: "57B0002600000001"

NUM_CONFIGURATIONS

public static final String NUM_CONFIGURATIONS = "USB.device.bNumConfigurations"

OPTIONAL property key. Value is "USB.device.bNumConfigurations". The value is Integer.

INTERFACE_NUMBER

public static final String INTERFACE_NUMBER = "USB.device.bInterfaceNumber"

MANDATORY property key. Value is "USB.device.bInterfaceNumber". The value is Integer.

ALTERNATE_SETTING

public static final String ALTERNATE_SETTING = "USB.device.bAlternateSetting"

OPTIONAL property key. Value is "USB.device.bAlternateSetting". The value is Integer.

NUM_ENDPOINTS

public static final String NUM_ENDPOINTS = "USB.device.bAlternateSetting"

OPTIONAL property key. Value is "USB.device.bAlternateSetting". The value is Integer.

INTERFACE_CLASS

public static final String INTERFACE_CLASS = "USB.device.bInterfaceClass"

MANDATORY property key. Value is "USB.device.bInterfaceClass". The value is String, hexadecimal, 2-digits. Example: "ff"
**INTERFACE_SUBCLASS**

public static final String INTERFACE_SUBCLASS = "USB.device.bInterfaceSubClass"

MANDATORY property key. Value is "USB.device.bInterfaceSubClass". The value is String, hexadecimal, 2-digits. Example: "ff"

**INTERFACE_PROTOCOL**

public static final String INTERFACE_PROTOCOL = "USB.device.bInterfaceProtocol"

MANDATORY property key. Value is "USB.device.bInterfaceProtocol". The value is String, hexadecimal, 2-digits. Example: "ff"

**INTERFACE_DESCRIPTION**

public static final String INTERFACE_DESCRIPTION = "USB.device.iInterface"

OPTIONAL property key. Value is "USB.device.iInterface". The value is String of indicated in iInterface. (The value is not the index.)

**USB_BUS**

public static final String USB_BUS = "USB.device.bus"

MANDATORY property key. Value is "USB.device.bus". The value is Integer. Used to identify USB devices with same VID / PID. The value is the ID of the USB bus assigned when connecting the USB device. USB bus ID is integer. The USB bus ID does not change while the USB device remains connected. Example: 3

**USB_ADDR**

public static final String USB_ADDR = "USB.device.address"

MANDATORY property key. Value is "USB.device.address". The value is Integer. Used to identify USB devices with same VID / PID. The value is the ID of the USB address assigned when connecting the USB device. USB address is integer (001-127). The USB address does not change while the USB device remains connected. Example: 2

**MATCH_MODEL**

public static final int MATCH_MODEL = 10

Constant for the USB device match scale, indicating a match with USB.device.idVendor and USB.device.idProduct. Value is 10.

**MATCH_PROTOCOL**

public static final int MATCH_PROTOCOL = 7
interface USBDevice

Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass, USB.device.bDeviceSubClass and USB.device.bDeviceProtocol, or a match with bInterfaceClass, bInterfaceSubClass and bInterfaceProtocol in one of USB.device.interfaceclasses. Value is 7.

MATCH_SUBCLASS

public static final int MATCH_SUBCLASS = 5

Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass and USB.device.bDeviceSubClass, or a match with bInterfaceClass and bInterfaceSubClass in one of USB.device.interfaceclasses. Value is 5.

MATCH_CLASS

public static final int MATCH_CLASS = 3

Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass, or a match with bInterfaceClass in one of USB.device.interfaceclasses. Value is 3.

DEVICE_CATEGORY_SERIAL

public static final String DEVICECATEGORY_SERIAL = "Serial"

OPTIONAL Property. The value is "Serial". Constant for the value of the service property DEVICE_CATEGORY used for a USB device which has a serial communication function such as a USB dongle. The USB base driver bundle must set this property key and serial.comport property. This device category's value may be used independently of USB. This value is defined because some USB devices have a serial communication function. See Also org.osgi.service.device.Constants.DEVICE_CATEGORY

COM_PORT

public static final String COM_PORT = "serial.comport"

OPTIONAL Property key. Value is "serial.comport". The property value is String. The USB Device has a serial communication function, set the value that represents the COM port. If the USB device does not have a serial communication function, this key and value are not set. The driver can communicate through Java Communications API with this value. Set this value "portName" of javax.comm.CommPortIdentifier#getPortIdentifier(String portName). Then serial communication is possible. Serial.comport value's format must be equal to the "portName" format. If a USB base driver set this property, USBDevice.DEVICECATEGORY_SERIAL must be set to DEVICECATEGORY. Example1: "/dev/ttyUSB0". Example2: "COM5". Example3: "/dev/tty.usbserial-XXXXXX".

DEVICE_CATEGORY_MASSSTORAGE

public static final String DEVICECATEGORY_MASSSTORAGE = "MassStorage"

OPTIONAL Property. The value is "MassStorage". Constant for the value of the service property DEVICE_CATEGORY used for a USB device which is a MassStorage Class in USB Specification such as a USB storage. Such a USB base driver bundle must set this property key and massstorage.mountpoints property while the device is mounted. See Also org.osgi.service.device.Constants.DEVICE_CATEGORY
MOUNTPOINTS

public static final String MOUNTPOINTS = "massstorage.mountpoints"

OPTIONAL Property key. Value is "massstorage.mountpoints". The property value is String[]. If the USB device is Mass Storage Class, set the value that represents the mount point (a path to the USB storage) in OS. If the USB device is not Mass Storage Class, this key and value are not set. The driver can read and write the USB storage through standard File API with this value. Set this value "pathname" of java.io.File(String pathname). Then file access is possible. Massstorage.mountpoints's format must be equal to the "pathname" format. If a USB base driver set this property, USBDevice.DEVICE_CATEGORY_MASSSTORAGE must be set to DEVICE_CATEGORY. Example1: {"/mnt/media/usb-storage-01"}. Example2: {"D:\Java"}. 

DocFlex/Doclet v1.5.6

DocFlex/Doclet is both a multi-format Javadoc doclet and a free edition of DocFlex/Javadoc. If you need to customize your Javadoc without writing a full-blown doclet from scratch, DocFlex/Javadoc may be the only tool able to help you! Find out more at www.docflex.com
8 Considered Alternatives

8.1 USB.device.interfaceclasses

The alternative format of USB.device.interfaceclasses is below.

- **USB.device.interfaceclasses** — MANDATORY property key. The property value is an array of `String` for each interface descriptor. Each `String` value is connected with "_" interface descriptor’s `bInterfaceClass`, `bInterfaceSubClass`, and `bInterfaceProtocol`. If there is no subclass code associated with the class code, does not connect subclass code and protocol code. If there is no protocol code associated with the class code, the protocol code is not connected. In addition, if the class code is vendor-specific class, does not connect subclass code and protocol code. Set only the class code. (See Table 6.)

  - Example: An example of a USB device that has 2 interfaces. The first class code is CDC, subclass code is ACM (without protocol code). The second class code is CDC-Data (no subclass code and protocol code).

    - Value: "CDC_ACM", "CDC-Data".

<table>
<thead>
<tr>
<th>Class Code</th>
<th>SubClass Code</th>
<th>Protocol Code</th>
<th>Representation in Service Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>any</td>
<td>Audio_AudioControl</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>any</td>
<td>Audio_AudioStreaming</td>
</tr>
<tr>
<td>3</td>
<td>01</td>
<td>any</td>
<td>Audio_MidiStreaming</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
<td>other than above</td>
<td>any</td>
</tr>
<tr>
<td>5</td>
<td>02</td>
<td>01</td>
<td>CDC_DLCM_V.250</td>
</tr>
<tr>
<td>6</td>
<td>02</td>
<td>other than above</td>
<td>any</td>
</tr>
<tr>
<td>7</td>
<td>02</td>
<td>02</td>
<td>CDC_ACM_V.250</td>
</tr>
<tr>
<td>8</td>
<td>02</td>
<td>other than above</td>
<td>any</td>
</tr>
<tr>
<td>9</td>
<td>02</td>
<td>03</td>
<td>CDC_TCM_V.250</td>
</tr>
<tr>
<td>10</td>
<td>02</td>
<td>other than above</td>
<td>any</td>
</tr>
<tr>
<td>11</td>
<td>02</td>
<td>04</td>
<td>CDC_MCCM</td>
</tr>
<tr>
<td>12</td>
<td>02</td>
<td>05</td>
<td>CDC_CAPI</td>
</tr>
<tr>
<td>13</td>
<td>02</td>
<td>06</td>
<td>CDC_ENCM</td>
</tr>
<tr>
<td>14</td>
<td>02</td>
<td>07</td>
<td>CDC_ATM</td>
</tr>
</tbody>
</table>

Table 6: Class Code and Service Property
<table>
<thead>
<tr>
<th>Octet 1</th>
<th>Octet 2</th>
<th>Octet 3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>08</td>
<td>02</td>
<td>CDC_WHCM_PCCA-101</td>
</tr>
<tr>
<td>02</td>
<td>08</td>
<td>03</td>
<td>CDC_WHCM_PCCA-101-AnnexO</td>
</tr>
<tr>
<td>02</td>
<td>08</td>
<td>04</td>
<td>CDC_WHCM_GSM7.07</td>
</tr>
<tr>
<td>02</td>
<td>08</td>
<td>05</td>
<td>CDC_WHCM_3GPP27.007</td>
</tr>
<tr>
<td>02</td>
<td>08</td>
<td>06</td>
<td>CDC_WHCM_TIA-CDMA</td>
</tr>
<tr>
<td>02</td>
<td>08</td>
<td>FE</td>
<td>CDC_WHCM_ExternalProtocol</td>
</tr>
<tr>
<td>02</td>
<td>08</td>
<td>other than above</td>
<td>CDC_WHCM</td>
</tr>
<tr>
<td>02</td>
<td>09</td>
<td>02</td>
<td>CDC_DM_PCCA-101</td>
</tr>
<tr>
<td>02</td>
<td>09</td>
<td>03</td>
<td>CDC_DM_PCCA-101-AnnexO</td>
</tr>
<tr>
<td>02</td>
<td>09</td>
<td>04</td>
<td>CDC_DM_GSM7.07</td>
</tr>
<tr>
<td>02</td>
<td>09</td>
<td>05</td>
<td>CDC_DM_3GPP27.007</td>
</tr>
<tr>
<td>02</td>
<td>09</td>
<td>06</td>
<td>CDC_DM_TIA-CDMA</td>
</tr>
<tr>
<td>02</td>
<td>09</td>
<td>other than above</td>
<td>CDC_DM</td>
</tr>
<tr>
<td>02</td>
<td>0A</td>
<td>02</td>
<td>CDC_MDLM_PCCA-101</td>
</tr>
<tr>
<td>02</td>
<td>0A</td>
<td>03</td>
<td>CDC_MDLM_PCCA-101-AnnexO</td>
</tr>
<tr>
<td>02</td>
<td>0A</td>
<td>04</td>
<td>CDC_MDLM_GSM7.07</td>
</tr>
<tr>
<td>02</td>
<td>0A</td>
<td>05</td>
<td>CDC_MDLM_3GPP27.007</td>
</tr>
<tr>
<td>02</td>
<td>0A</td>
<td>06</td>
<td>CDC_MDLM_TIA-CDMA</td>
</tr>
<tr>
<td>02</td>
<td>0A</td>
<td>FE</td>
<td>CDC_MDLM_ExternalProtocol</td>
</tr>
<tr>
<td>02</td>
<td>0A</td>
<td>other than above</td>
<td>CDC_MDLM</td>
</tr>
<tr>
<td>02</td>
<td>0B</td>
<td>02</td>
<td>CDC_OBEX_PCCA-101</td>
</tr>
<tr>
<td>02</td>
<td>0B</td>
<td>03</td>
<td>CDC_OBEX_PCCA-101-AnnexO</td>
</tr>
<tr>
<td>02</td>
<td>0B</td>
<td>04</td>
<td>CDC_OBEX_GSM7.07</td>
</tr>
<tr>
<td>02</td>
<td>0B</td>
<td>05</td>
<td>CDC_OBEX_3GPP27.007</td>
</tr>
<tr>
<td>02</td>
<td>0B</td>
<td>06</td>
<td>CDC_OBEX_TIA-CDMA</td>
</tr>
<tr>
<td>02</td>
<td>0B</td>
<td>FE</td>
<td>CDC_OBEX_ExternalProtocol</td>
</tr>
<tr>
<td>02</td>
<td>0B</td>
<td>other than above</td>
<td>CDC_OBEX</td>
</tr>
<tr>
<td>02</td>
<td>0C</td>
<td>02</td>
<td>CDC_EEM_EEM</td>
</tr>
<tr>
<td>02</td>
<td>0C</td>
<td>other than above</td>
<td>CDC_EEM</td>
</tr>
<tr>
<td>02</td>
<td>0D</td>
<td>FE</td>
<td>CDC_NCM_ExternalProtocol</td>
</tr>
<tr>
<td>02</td>
<td>0D</td>
<td>other than above</td>
<td>CDC_NCM</td>
</tr>
<tr>
<td>02</td>
<td>other than above</td>
<td>any</td>
<td>CDC</td>
</tr>
<tr>
<td>02</td>
<td>01</td>
<td>01</td>
<td>HID_Boot_KeyBoard</td>
</tr>
<tr>
<td>02</td>
<td>01</td>
<td>02</td>
<td>HID_Boot_Mouse</td>
</tr>
<tr>
<td>02</td>
<td>other than above</td>
<td>any</td>
<td>HID</td>
</tr>
<tr>
<td>05</td>
<td>any</td>
<td>any</td>
<td>Physical</td>
</tr>
<tr>
<td>06</td>
<td>01</td>
<td>01</td>
<td>Image_Capture_PIAM15740</td>
</tr>
<tr>
<td>06</td>
<td>01</td>
<td>other than above</td>
<td>Image_Capture</td>
</tr>
<tr>
<td>06</td>
<td>other than above</td>
<td>any</td>
<td>Image</td>
</tr>
<tr>
<td>07</td>
<td>01</td>
<td>01</td>
<td>Printer_Printer_Unidirectional</td>
</tr>
<tr>
<td>07</td>
<td>01</td>
<td>02</td>
<td>Printer_Printer_Bi-directional</td>
</tr>
<tr>
<td>07</td>
<td>01</td>
<td>03</td>
<td>Printer_Printer_1284.4</td>
</tr>
<tr>
<td>07</td>
<td>01</td>
<td>FF</td>
<td>Printer_Printer_VendorSpecific</td>
</tr>
<tr>
<td>07</td>
<td>01</td>
<td>other than above</td>
<td>Printer</td>
</tr>
<tr>
<td>08</td>
<td>00</td>
<td>any</td>
<td>MassStorage_SCSI-nr</td>
</tr>
<tr>
<td>08</td>
<td>01</td>
<td>any</td>
<td>MassStorage_RBC</td>
</tr>
<tr>
<td>Device ID</td>
<td>Instance</td>
<td>Subclass</td>
<td>Class</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>64 08 02</td>
<td>00</td>
<td>MassStorage_MMC-5 CBI-cci</td>
<td></td>
</tr>
<tr>
<td>65 08 02</td>
<td>01</td>
<td>MassStorage_MMC-5 CBI-ncci</td>
<td></td>
</tr>
<tr>
<td>66 08 02</td>
<td>50</td>
<td>MassStorage_MMC-5 BBB</td>
<td></td>
</tr>
<tr>
<td>67 08 02</td>
<td>other than above</td>
<td>MassStorage_MMC-5</td>
<td></td>
</tr>
<tr>
<td>68 08 04</td>
<td>00</td>
<td>MassStorage_UFI CBI-cci</td>
<td></td>
</tr>
<tr>
<td>69 08 04</td>
<td>01</td>
<td>MassStorage_UFI CBI-ncci</td>
<td></td>
</tr>
<tr>
<td>70 08 04</td>
<td>other than above</td>
<td>MassStorage_UFI</td>
<td></td>
</tr>
<tr>
<td>71 08 06</td>
<td>50</td>
<td>MassStorage_SCSI BBB</td>
<td></td>
</tr>
<tr>
<td>72 08 06</td>
<td>62</td>
<td>MassStorage_SCSI UAS</td>
<td></td>
</tr>
<tr>
<td>73 08 06</td>
<td>other than above</td>
<td>MassStorage_SCSI</td>
<td></td>
</tr>
<tr>
<td>74 08 07</td>
<td>any</td>
<td>MassStorage_LSDFS</td>
<td></td>
</tr>
<tr>
<td>75 08 08</td>
<td>any</td>
<td>MassStorage_IEEE1667</td>
<td></td>
</tr>
<tr>
<td>76 08 FF</td>
<td>00</td>
<td>MassStorage_VendorSpecific CBI-cci</td>
<td></td>
</tr>
<tr>
<td>77 08 FF</td>
<td>01</td>
<td>MassStorage_VendorSpecific CBI-ncci</td>
<td></td>
</tr>
<tr>
<td>78 08 FF</td>
<td>50</td>
<td>MassStorage_VendorSpecific BBB</td>
<td></td>
</tr>
<tr>
<td>79 08 FF</td>
<td>other than above</td>
<td>MassStorage_VendorSpecific</td>
<td></td>
</tr>
<tr>
<td>80 08 other than above</td>
<td>any</td>
<td>MassStorage</td>
<td></td>
</tr>
<tr>
<td>81 0A 00</td>
<td>01</td>
<td>CDC-Data_Data NTB</td>
<td></td>
</tr>
<tr>
<td>82 0A 00</td>
<td>30</td>
<td>CDC-Data_Data I.430</td>
<td></td>
</tr>
<tr>
<td>83 0A 00</td>
<td>31</td>
<td>CDC-Data_Data HDLC</td>
<td></td>
</tr>
<tr>
<td>84 0A 00</td>
<td>32</td>
<td>CDC-Data_Data Transparent</td>
<td></td>
</tr>
<tr>
<td>85 0A 00</td>
<td>50</td>
<td>CDC-Data_Data Q.921M</td>
<td></td>
</tr>
<tr>
<td>86 0A 00</td>
<td>51</td>
<td>CDC-Data_Data Q.921</td>
<td></td>
</tr>
<tr>
<td>87 0A 00</td>
<td>52</td>
<td>CDC-Data_Data Q.921TM</td>
<td></td>
</tr>
<tr>
<td>88 0A 00</td>
<td>90</td>
<td>CDC-Data_Data V.42bis</td>
<td></td>
</tr>
<tr>
<td>89 0A 00</td>
<td>91</td>
<td>CDC-Data_Data Euro-ISDN</td>
<td></td>
</tr>
<tr>
<td>90 0A 00</td>
<td>92</td>
<td>CDC-Data_Data V.120</td>
<td></td>
</tr>
<tr>
<td>91 0A 00</td>
<td>93</td>
<td>CDC-Data_Data CAPI2.0</td>
<td></td>
</tr>
<tr>
<td>92 0A 00</td>
<td>FD</td>
<td>CDC-Data_Data HostBasedDriver</td>
<td></td>
</tr>
<tr>
<td>93 0A 00</td>
<td>FE</td>
<td>CDC-Data_Data PUFD</td>
<td></td>
</tr>
<tr>
<td>94 0A 00</td>
<td>FF</td>
<td>CDC-Data_Data VendorSpecific</td>
<td></td>
</tr>
<tr>
<td>95 0A 00</td>
<td>other than above</td>
<td>CDC-Data_Data</td>
<td></td>
</tr>
<tr>
<td>96 0A other than above</td>
<td>any</td>
<td>CDC-Data</td>
<td></td>
</tr>
<tr>
<td>97 0B 00</td>
<td>00</td>
<td>SmartCard_SmartCard CCID</td>
<td></td>
</tr>
<tr>
<td>98 0B 00</td>
<td>01</td>
<td>SmartCard_SmartCard ICC-A</td>
<td></td>
</tr>
<tr>
<td>99 0B 00</td>
<td>02</td>
<td>SmartCard_SmartCard ICC-B</td>
<td></td>
</tr>
<tr>
<td>100 0B 00</td>
<td>other than above</td>
<td>SmartCard_SmartCard</td>
<td></td>
</tr>
<tr>
<td>101 0B other than above</td>
<td>any</td>
<td>SmartCard</td>
<td></td>
</tr>
<tr>
<td>102 0D 00</td>
<td>00</td>
<td>ContentSecurity</td>
<td></td>
</tr>
<tr>
<td>103 0E 01</td>
<td>any</td>
<td>Video_VideoControl</td>
<td></td>
</tr>
<tr>
<td>104 0E 02</td>
<td>any</td>
<td>Video_VideoStreaming</td>
<td></td>
</tr>
<tr>
<td>105 0E 03</td>
<td>any</td>
<td>Video_VideoInterfaceCollection</td>
<td></td>
</tr>
<tr>
<td>106 0E other than above</td>
<td>any</td>
<td>Video</td>
<td></td>
</tr>
<tr>
<td>107 0F any</td>
<td>any</td>
<td>PersonalHealthcareDevice</td>
<td></td>
</tr>
<tr>
<td>108 10 01</td>
<td>any</td>
<td>AudioVideoDevice_AVControlInterface</td>
<td></td>
</tr>
<tr>
<td>109 10 02</td>
<td>any</td>
<td>AudioVideoDevice_AVDataVideoStreamingInterface</td>
<td></td>
</tr>
<tr>
<td>110 10 03</td>
<td>any</td>
<td>AudioVideoDevice_AVDataAudioStreamingInterface</td>
<td></td>
</tr>
<tr>
<td>111 10 other than above</td>
<td>any</td>
<td>AudioVideoDevice</td>
<td></td>
</tr>
</tbody>
</table>
### Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented. `ServicePermission` is needed when a bundle get `USBDevice` service.
10 Document Support

10.1 References


10.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Yukio KOIKE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>NTT Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>1-1, Hikari-no-oka, Yokosuka-shi, 238-0847, Kanagawa, Japan</td>
</tr>
<tr>
<td>Voice</td>
<td>+81 46 859 5142</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:koike.yukio@lab.ntt.co.jp">koike.yukio@lab.ntt.co.jp</a></td>
</tr>
</tbody>
</table>

10.3 Acronyms and Abbreviations

10.4 End of Document
Abstract

The Remote Service Admin specification is lacking a mechanism to notify consumers of changes to an endpoint. The EndpointListener interface defines endpointAdded and endpointRemoved callbacks, but no mechanism to convey that an endpoint has been modified, for example because the service properties of the backing service have changed. This RFC addresses this issue.
0  Document Information

0.1  License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

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

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0 Document Information.................................................................................................................................................2

0.1 License.................................................................................................................................................................2

0.2 Trademarks..............................................................................................................................................................3

0.3 Feedback.................................................................................................................................................................3

0.4 Table of Contents......................................................................................................................................................3

0.5 Terminology and Document Conventions............................................................................................................4

0.6 Revision History..........................................................................................................................................................4

1 Introduction...............................................................................................................................................................4

2 Application Domain.......................................................................................................................................................5

3 Problem Description.......................................................................................................................................................5

4 Requirements...............................................................................................................................................................5

5 Technical Solution.........................................................................................................................................................5

6 Data Transfer Objects....................................................................................................................................................6

7 Javadoc.........................................................................................................................................................................6

8 Considered Alternatives..............................................................................................................................................6
0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

0.6 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>August, 2013</td>
<td>David Bosschaert, initial version of separate RFC. Previous design has been taking place in RFC 183.</td>
</tr>
<tr>
<td>0.1</td>
<td>September, 2013</td>
<td>Post September F2F discussion - add the new &quot;update&quot; methods to the ExportRegistration and ImportRegistration</td>
</tr>
<tr>
<td>0.2</td>
<td>September, 2013</td>
<td>Clarify the uniqueness requirements for endpoint ids, include community feedback</td>
</tr>
<tr>
<td>0.3</td>
<td>October, 2013</td>
<td>Update based on comments from 23/10 EEG call. Restructure sections to place background information in section 3. No changes to content.</td>
</tr>
<tr>
<td>0.4</td>
<td>November, 2013</td>
<td>Remove unused sections</td>
</tr>
<tr>
<td>0.5</td>
<td>December, 2013</td>
<td>Update Namespaces to separate out the extender section. Use three separate namespaces for the RSA actor capabilities</td>
</tr>
<tr>
<td>0.6</td>
<td>February, 2014</td>
<td>Update JavaDoc post Austin F2F. Minor changes to address bugs raised by RI and CT teams</td>
</tr>
<tr>
<td>0.7</td>
<td>March, 2014</td>
<td>Add clarifications as to the required property update support required from distribution providers</td>
</tr>
<tr>
<td>0.8</td>
<td>March, 2014</td>
<td>Update the &quot;update&quot; section to describe why shared ExportRegistrations are being removed from the spec</td>
</tr>
</tbody>
</table>
1 Introduction

The OSGi Remote Services and Remote Service Admin specifications describe how OSGi services can be remoted and how to consume these remote services using the OSGi Services programming model.

The Remote Service Admin specification version 1.0 defines how listeners are notified of endpoints being added and removed. However, the associated API does not support notifying listeners of changes to endpoints such as service property changes of the associated service. This RFC addresses this issue by proposing an extension to the Remote Service Admin specification.

2 Application Domain

This RFC relates to the domain of remote OSGi Services, specifically the Remote Service Admin specification.

3 Problem Description

In Remote Service Admin 1.0 it was possible for Topology Managers to dynamically register and unregister remote services, however there was no way to update a service without unregistering it. This causes significant disruption when remote clients are dependent upon the availability of the remote service for their operation.

It has also been found that part of the Remote Service Admin 1.0 specification wording relating to Endpoint Id uniqueness is insufficiently clear, in that it does not specify the scope of the uniqueness. This needs to be clarified so that different Topology Managers, Discovery Providers and Distribution Providers can reliably interoperate.

3.1 Notifying the Discovery Provider of updates

The EndpointListener interface is used to implement a distributed discovery mechanism and it allows the registration of a listener for distributed endpoints to appear and disappear via the endpointAdded() and endpointRemoved() callback methods. However, an endpoint can also change. This is in particular the case when the service registration properties of the endpoint are modified. Such modifications are not supported by the EndpointListener today, it sends a sequence of endpointRemoved() and endpointAdded() callbacks in such a case which can cause unnecessary volatility in the system.
### 3.2 Notifying the Distribution Provider of updates

In the Remote Service Admin specification the Topology Manager is responsible for interacting with RSA distribution providers. In RSA 1.0 this was broadly limited to creating and closing ImportRegistration and ExportRegistration objects using the Distribution Provider. To support updates RSA 1.1 will need additional interaction mechanisms to indicate that an ImportRegistration or ExportRegistration should be updated. It is necessary for the Topology Manager to initiate these updates, because either:

- It is then responsible for notifying Discovery services of any changes to an exported endpoint

or

- The topology manager is the component that is notified of changes to a remote endpoint via Discovery announcements.

### 3.3 Clarifying Endpoint Id Uniqueness

The following is an extract from the OSGi R5 compendium:

**122.4.3 Endpoint Id**

An Endpoint Id is an opaque unique identifier for an Endpoint. There is no syntax defined for this string except that white space at the beginning and ending must be ignored. The actual syntax for this Endpoint Id must be defined by the actual configuration type.

Two Endpoint Descriptions are deemed identical when their Endpoint Id is equal. The Endpoint Ids must be compared as string compares with leading and trailing spaces removed. The Endpoint Description class must use the String class' hash Code from the Endpoint Id as its own hashCode.

Furthermore the OSGi R5 compendium states that:

Two Endpoint Descriptions are deemed equal when their Endpoint Id is equal. The Endpoint Id is a mandatory property of an Endpoint Description, it is further described at Endpoint Id on page 709. The hash code is therefore also based on the Endpoint Id.

**122.4.1 Validity**

A valid Endpoint Description must at least satisfy the following assertions:

- It must have a non-null Id that uniquely identifies the Endpoint

The extracts above are sufficient to require that:

1. The Endpoint Id is opaque, and has no declared format or syntax

2. The Endpoint Id defines the identity of an EndpointDescription, regardless of its other properties

3. The Endpoint Id is expected to be “unique”, although the scope of this uniqueness is not expressly defined.

### 3.3.1 Required Scope of Endpoint Id Uniqueness

The Remote Service Admin specification states that there may be multiple Topology Managers, Distribution Providers and Discovery Providers active concurrently within a single framework. It is clear that if these implementations are to coexist then Endpoint Id uniqueness must hold within the OSGi framework. This applies
regardless of the number of installed Distribution Providers, more than one of which may be exporting a given service.

In the Remote Service Admin Specification EndpointDescriptions are created by distribution providers, but are passed on to Topology Managers. Depending upon the Topology Manager's implementation these EndpointDescription objects may then be passed to other actors, such as Discovery Providers, via EndpointListener or EndpointEventListener services. Discovery providers then advertise EndpointDescription objects over the network. These advertisements result in the EndpointDescription being serialized and reconstituted on a remote machine.

The above scenario adds some constraints on the uniqueness of EndpointDescription Ids.

1. Portable Discovery Providers can only use the EndpointDescription Id to determine which endpoint is being announced, updated or revoked. This applies both when being notified of local and remote events.

2. Portable Topology Managers can only use the EndpointDescription Id to determine which EndpointDescription they are receiving an event for.

As EndpointDescription objects are made available remotely, and therefore shared between frameworks, the required scope of Id uniqueness required is larger than a single framework.

If three frameworks are connected by a Discovery provider, and two produce an EndpointDescription with the same Id, then two "identical" notification events will arrive at the third framework. If one of the two frameworks then destroys the endpoint, and advertises the service removal to the third framework, then the third framework will reach the incorrect conclusion that there are now zero endpoints available.

We can therefore state that the minimum scope of Endpoint Id uniqueness is that no two distinct endpoints should have the same Id within a connected group. Distribution Providers must ensure that they do not produce Endpoint Id clashes within a connected group.

### 3.3.2 Ensuring Endpoint Ids are sufficiently Unique

It should be noted that Discovery Providers can be added to a framework at any time, increasing the size of a connected group. New Distribution Providers that can support additional configuration types can also be added dynamically, as can Topology Managers with the ability to source EndpointDescriptions from XML, representing external services. This means that although an Endpoint Id must only be unique with a single connected group, the group can expand in size or number of EndpointDescription objects representing a given service at any time.

### 3.4 Make Event property names consistent

In section 122.7.1 of RSA 1.0 the specification defined property mappings for mapping RemoteServiceAdminEvent objects to OSGi Event Admin events. This includes the following:

- **cause** – The exception, if present.

The constant “cause” does not match the normal pattern used by Event Admin, which would normally use the String “exception” as a key. There is also no constant defined by Remote Service Admin for “cause”.

### 3.5 Define capability namespaces for RSA components

The Remote Services Admin specification defines multiple actors. Discovery providers, Distribution Providers and Topology Managers. There is also an extender defined for consuming “local” XML endpoint descriptions from inside bundles.
These actors should provide well-defined capabilities so that they can be easily located in a repository, and to enable simpler provisioning of implementations. The local discovery extender should also define a capability in the osgi.extender namespace.

4 Requirements

RSA01 – The Solution MUST define a mechanism to provide Endpoint Listeners with a notification when an endpoint was modified.

RSA02 – The Solution SHOULD allow a Topology Manager to update the service properties an Exported or Imported Service without unregistering it. This may not be possible if the configuration type or access intent of the service changes.

5 Technical Solution

5.1 Discovering Remote Service Updates
To receive modification events a new EndpointEventListener interface can be implemented by the listener. The EndpointEventListener follows a similar pattern as the ServiceListener in the core framework. The event holds a type attribute describing the type of event.

5.1.1 Backward compatibility
The existing EndpointListener interface sends a endpointRemoved() callback followed by an endpointAdded() callback in case an endpoint registration has changed (e.g. properties added or removed). The EndpointListener interface will continue to behave this way.

5.1.2 EndpointEventListener
The new EndpointEventListener will not send a sequence of REMOVED and ADDED events in such a case, but rather send a single MODIFIED or MODIFIED_ENDMATCH event, whichever is appropriate.

The EndpointEventListener is defined as follows:

```java
public interface EndpointEventListener {
    void endpointChanged(EndpointEvent event, String filter);
}
```
public class EndpointEvent {
    public static final int ADDED = 0x00000001;
    public static final int REMOVED = 0x00000002;
    public static final int MODIFIED = 0x00000004;
    public static final int MODIFIED_ENDMATCH = 0x00000008;

    private final EndpointDescription endpoint;
    private final int type;

    public EndpointEvent(int type, EndpointDescription endpoint) {
        super(endpoint);
        this.endpoint = endpoint;
        this.type = type;
    }

    public EndpointDescription getEndpoint() {
        return endpoint;
    }

    public int getType() {
        return type;
    }
}

5.2 Updating Exported and Imported Services

To support requirement RSA02 cases it is necessary to add update methods to both ImportRegistration and ExportRegistration. As these are “provider types” that should only be implemented by RSA Distribution providers this represents a minor change to the RSA API.

Remote Service updates are intended to address the situation where one or more service properties have changed on a remotable service without making it appear as though the service has become unavailable. This requires that the exporting distribution provider and the importing distribution provider support the changing of properties. In the RSA specification there are two types of service properties. The first type are properties that are intended to be consumed by the distribution provider, such as:

- The exported configuration types
- The exported interfaces
- configuration type specific properties
- exported intents

We will refer to the second type of service properties as “general” service properties. These are typically used to communicate information to the consumer of the service, and not interpreted by the distribution provider.

The Remote Service Admin 1.1 specification requires that distribution providers MUST support the update of general service properties, both when exporting and importing. Distribution Providers MAY fail an update of a general service property if its value is not one of the OSGi scalar types. Distribution providers MAY also support
updates to other properties. If an unsupported property update is requested then the distribution provider MUST fail the export/import operation.

5.2.1 ExportRegistration

The ExportRegistration represents an OSGi service that has been exported by a Distribution Provider. It is the Topology Manager's role to track and manage which services are exported, therefore an update method should be added to this interface. The Topology Manager should also have an opportunity to change the set of properties that were passed when originally creating the ExportRegistration.

```java
/**
 * Update the endpoint represented by this {@link ExportRegistration} and
 * return an updated {@link EndpointDescription}. If this method returns an
 * updated {@link EndpointDescription}, then the object returned via
 * {@link #getExportReference()} must also have been updated to return this
 * new object. If this method does not return an updated
 * {@link EndpointDescription} then the object returned via
 * {@link #getExportReference()} should remain unchanged.
 *
 * When creating the updated {@link EndpointDescription} the
 * {@link ServiceReference} originally passed to
 * {@link RemoteServiceAdmin#exportService(ServiceReference, Map)} must be
 * queried to pick up any changes to its service properties.
 *
 * If this argument is null then the original properties passed when
 * creating this ExportRegistration should be used when constructing the
 * updated {@link EndpointDescription}. Otherwise the new properties should
 * be used, and replace the original properties for subsequent calls to the
 * update method.
 *
 * @param properties properties to be merged with the current service
 * properties for the {@link ServiceReference} represented by this
 * {@link ExportRegistration}. If null is passed then the original
 * properties passed to
 * {@link RemoteServiceAdmin#exportService(ServiceReference, Map)}
 * will be used.
 * @return The updated {@link EndpointDescription} for this registration or
 * null if there was a failure updating the endpoint. If a failure
 * occurs then it can be accessed using {@link #getException()}.
 * @throws IllegalStateException If this registration is closed, or when
 * this registration was not properly initialized. See
 * {@link #getException()}.
 *
 * @since 1.1
 */
EndpointDescription update(Map<String, ?> properties);
```

5.2.2 ImportRegistration

The ExportRegistration represents an OSGi service that has been imported into the framework by a Distribution Provider. It is the Topology Manager's role to track and manage which services are imported, but not to actually register the imported service, therefore an update method should be added to this interface. The Topology Manager must be able to pass the new EndpointDescription for this imported service. When an ImportRegistration
is updated it MUST NOT create a new ServiceRegistration for the endpoint, i.e. the local service registry should experience no interruption of service, and the updated service should have the same service id before and after the update. If this is not possible then the update MUST fail.

```java
/**
 * Update the local service represented by this {@link ImportRegistration}.
 * After this method returns the {@link EndpointDescription} returned via
 * {@link #getImportReference()} must have been updated.
 *
 * @param endpoint The updated endpoint
 *
 * @return <code>true</code> if the endpoint was successfully updated,
 * <code>false</code> otherwise. If the update fails then the
 * failure can be retrieved from {@link #getException()}.
 *
 * @throws IllegalStateException When this registration is closed, or if it
 * was not properly initialized. See {@link #getException()}.  
 * @throws IllegalArgumentException When the supplied
 * {@link EndpointDescription} does not represent the same endpoint
 * as this {@link ImportRegistration}.
 *
 * @since 1.1
 */
boolean update(EndpointDescription endpoint);
```

### 5.2.3 Update events for RemoteServiceAdmin

The Remote Service Admin specification defines RemoteServiceAdminEvent and RemoteServiceAdminEventListener types which are used to provide notifications when Endpoints are added and removed. This RFC defines two new event types, one for ExportRegistration update, and one for ImportRegistration update. These events should be issued when the relevant update succeeds. If an update fails either with an error or a warning, then the relevant error or warning event should be issued.

### 5.2.4 Shared Endpoints and updates

Version 1.0 of the Remote Service Admin specification contained the concept of "linked" registrations both for ImportRegistrations and ExportRegistrations. These were added to optimise the creation of remote endpoints and local proxies in the case where multiple topology managers exported or imported the same service with the same properties. This sharing complicates the life-cycle of Import and Export Registrations because they must not actually be closed until all Topology Managers have closed their copies.

This shared lifecycle further complicates the behaviour of remote service updates. If two Topology Managers share the same underlying ExportRegistration and only one wishes to update the service then the Distribution provider is unable to perform the update. Either one topology manager’s Export would be silently updated behind the scenes, or the other topology manager would need a new endpoint to be created. The same issue affects ImportRegistration objects, where the underlying ServiceRegistration must either be updated or duplicated.

In both cases this breaks the RSA update use-case. Therefore from RSA 1.1 this RFC proposes to remove the sections describing registration sharing. Separate Topology managers will now receive their own separate Endpoints. This adds a small overhead in the case where there are multiple topology managers that add identical properties, but this is a relatively niche usage. Typically there is only one topology manager, and if there is more than one it typically adds additional property information.
From a client perspective there will be almost no visible change. The one possible change is that in situations where the client previously saw one local service representing a remote endpoint they may now see more than one. The fact that these services represent the same remote service is easy to determine using the remote framework id and the remote service id. It should be noted that this case also occurs in RSA 1.0 when the same service is made available through multiple distribution providers. As a result client code must already be able to cope with this situation, and it does not represent a breaking change.

5.3 Clarify the uniqueness of EndpointDescription Id Strings

The RSA 1.1 specification should clarify the uniqueness of EndpointDescription Id Strings to state that they are required to be unique within the connected set of frameworks. This means that no two distribution providers connected by a discovery service should produce the same Id String unless the EndpointDescription is for the same endpoint.

5.3.1 Producing a suitably Unique identifier

The simplest way to ensure that a growth in the number of EndpointDescriptions and/or the size of the connected group does not violate the required uniqueness of Endpoint Ids is for implementations to make their Endpoint Ids globally unique. This protects against clashes regardless of changes to the connected group.

Whilst globally unique identifiers are a simple solution to the Endpoint Id uniqueness problem, they are not easy to implement in all environments. In some systems they can be prohibitively expensive to create, or of insufficient entropy to be genuinely unique. Some distribution providers may therefore choose not to use random globally unique ids.

In the case where no globally unique value is used the following actions are recommended (although not required).

1. Distribution Providers protect against intra-framework clashes using some known value unique to the service, for example the service id.

2. Distribution Providers protect against inter-provider collisions within a single framework by using some unique value, such as the distribution provider's bundle id. The distribution provider bundle's symbolic name is insufficient, as there may be multiple versions of the same distribution provider installed within a single framework.

3. Distribution Providers protect against inter-framework collisions using some value unique to the framework, such as the framework UUID.

These suggestions may be included in the specification, but are not intended to be normative, and no implementation should rely on certain values being contained within the id. Distribution Providers are free to generate Endpoint Id in any way, as long as it meets the required level of uniqueness.

5.4 Event Admin property mapping

The event mappings defined in Remote Service Admin 1.0 were missing constant definitions, and also inconsistent with normal mappings. To remedy this this RFC proposes that:

- Implementations continue to map failure exceptions to the “cause” property. This ensures backward compatibility between versions of the specification. There will continue to be no constant defined for this key.

- RSA 1.1 Implementations must also map failure exceptions to the “exception” property defined by Event Admin. This is expected to be the normal location that event receivers will use to find exception values.
5.5 Generic Capability namespaces

This RFC suggests that implementations make use of Generic Capabilities to describe their interaction within the Remote Services Admin specification.

5.5.1 The Local Discovery Extender

The RSA specification defines an extender mechanism for local discovery of EndpointDescription XML files. Currently there is no defined mechanism for EndpointDescription bundles to declare that they need this extender to be present. RSA 1.1 therefore defines the value 'osgi.remoteserviceadmin' for the 'osgi.extender' capability. This capability should be provided by Local Discovery extenders, and required by bundles that provide EndpointXML descriptions to the extender. The version of the provided capability is defined to be 1.1, to match the version of the specification that defined the capability.

Example capability for an implementation of the extender:

```
Provide-Capability: osgi.extender; osgi.extender=osgi.remoteserviceadmin; version:Version=1.1
```

Example requirement from a bundle that contains Endpoint XML files:

```
Require-Capability: osgi.extender; filter:=(&(osgi.extender=osgi.remoteserviceadmin) (version>=1.1)); effective:=active
```

5.5.2 The Discovery Provider Capability

Discovery Providers use the 'osgi.remoteserviceadmin.discovery' namespace to declare themselves as a discovery provider. The version defined for this namespace is version 1.1, and indicates that the discovery provider meets the RSA 1.1 specification requirements for discovery providers.

For Discovery providers there is an additional defined attribute, 'protocols'. The protocols attribute is of type List<String> and contains a list of the discovery protocols supported by the discovery provider. Local discovery providers (using the Endpoint XML extender), should use the value 'local' to indicate that they support local discovery. Other values for the protocols attribute are implementation specific.

The primary motivation for the 'osgi.remoteserviceadmin.discovery' capability is to allow for provisioning from OSGi Bundle Repositories using the Repository API. Bundles that expose remote services should not state a requirement for a Topology Manager capability.

Example: A discovery provider that provides local and SLP discovery:

```
Provide-Capability: osgi.remoteserviceadmin.discovery;protocols:List<String>="SLP,local"; version:Version=1.1
```

5.5.3 The Distribution Provider capability

Remote Service Admin distribution providers advertise their supported distribution mechanisms at runtime using configurations. These are selected at runtime using the service.exported.configs service property. The Generic Capability for distribution providers therefore defines a namespace 'osgi.remoteserviceadmin.distribution' with an attribute "configs", of type List<String>, that a distribution provider can use to advertise the configs that it supports. The version defined for this namespace is version 1.1, and indicates that the distribution provider meets the RSA 1.1 specification requirements for distribution providers.

The primary motivation for the 'osgi.remoteserviceadmin.distribution' capability is to allow for provisioning from OSGi Bundle Repositories using the Repository API. Bundles that expose remote services should not state a requirement for a Distribution Provider capability.
Example: A Distribution provider that provides JAX-WS and JAX-RS as distribution mechanisms:

```
Provide-Capability: osgi.remoteserviceadmin.distribution; configs:List<String>="jax-ws,jax-rs"; version:Version=1.1
```

5.5.4 The Topology Manager Capability

Remote Service Admin topology managers may use different policies when determining which services to import. The Generic capability namespace for topology managers is 'osgi.remoteserviceadmin.topology' and therefore includes the `policy` attribute. This attribute has a String value that may be one of the specification suggested policies, such as 'promiscuous' or 'fail-over', or a custom mechanism. The version defined for this namespace is version 1.1, and indicates that the distribution provider meets the RSA 1.1 specification requirements for topology managers.

The primary motivation for the 'osgi.remoteserviceadmin.topology' capability is to allow for provisioning from OSGi Bundle Repositories using the Repository API. Bundles that expose remote services should not state a requirement for a Topology Manager capability.

Example: A Topology Manager that uses the promiscuous policy

```
Provide-Capability: osgi.remoteserviceadmin.topology; policy=promiscuous; version:Version=1.1
```

6 Data Transfer Objects

The updates defined in this RFC do not require any additional DTOs

7 Considered Alternatives

7.1.1 The osgi.remoteserviceadmin namespace

The osgi.remoteserviceadmin namespace uses the `osgi.remoteserviceadmin` attribute to indicate the type of actor that the bundle is. There are three defined values for the `osgi.remoteserviceadmin` attribute: 'discovery', 'distribution' and 'topology'. The initial version of the osgi.remoteserviceadmin namespace is 1.1, to match the version of the RSA specification that introduced it, and is declared using the `version` attribute, of type Version.
Discovery Providers

For Discovery providers there is an additional defined attribute, 'protocols'. The protocols attribute is of type List<String> and contains a list of the discovery protocols supported by the discovery provider. Local discovery providers (using the Endpoint XML extender), should use the value 'local' to indicate that they support local discovery. Other values for the protocols attribute are implementation specific.

For consistency, Local Discovery extenders should also declare their extender capability using the osgi.extender namespace and the attribute value 'osgi.remoteserviceadmin'. The extender capability should be declared at version 1.1

Example: A discovery provider that provides local and SLP discovery:

Provide-Capability: osgi.extender; osgi.extender=osgi.remoteserviceadmin; version:Version=1.1, osgi.remoteserviceadmin=osgi.remoteserviceadmin=discovery; protocols:List<String>="SLP,local"; version:Version=1.1

Distribution Providers

Remote Service Admin distribution providers advertise their supported distribution mechanisms using configurations. These are selected at runtime using the service.exported.configs service property. The Generic Capability for distribution providers therefore defines an attribute "configs", of type List<String>, that a distribution provider can use to advertise the configs that it supports.

Example: A Distribution provider that provides JAX-WS and JAX-RS as distribution mechanisms:

Provide-Capability: osgi.remoteserviceadmin; osgi.remoteserviceadmin=distribution; configs:List<String>="jax-ws,jax-rs"; version:Version=1.1

Topology Managers

Remote Service Admin topology managers may use different policies when determining which services to import. The Generic capability for topology managers should therefore include the policy attribute. This attribute has a String value that may be one of the specification suggested policies, such as promiscuous or fail-over, or a custom mechanism.

Example: A Topology Manager that uses the promiscuous policy

Provide-Capability: osgi.remoteserviceadmin; osgi.remoteserviceadmin=topology; policy=promiscuous; version:Version=1.1

8 Security Considerations

The proposed changes in the RFC do not require any security changes
9 Document Support

9.1 References


9.2 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></td>
</tr>
<tr>
<td>Voice</td>
<td></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>Richard Nicholson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Paremus</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Voice</td>
<td></td>
</tr>
<tr>
<td>e-mail</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Tim Ward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Paremus</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:tim.ward@paremus.com">tim.ward@paremus.com</a></td>
</tr>
</tbody>
</table>
9.3 Acronyms and Abbreviations

9.4 End of Document
RFC 206 - Asynchronous Services

Draft

51 Pages

Abstract

10 point Arial Centered.

The OSGi service registry is used by bundles to collaborate using loosely coupled services, registered with one or more public interfaces that can be called. OSGi services are, like most Java objects, normally designed to be called synchronously. There are, however, often significant advantages that can be realized by clients when they execute one or more parts of their operation asynchronously. This RFC provides a generic mechanism that allows existing OSGi services with a synchronous API to be invoked asynchronously without requiring them to be modified.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0 Document Information .............................................................................................................................................. 2
   0.1 License .................................................................................................................................................. 2
   0.2 Trademarks ........................................................................................................................................... 3
   0.3 Feedback ............................................................................................................................................. 3
   0.4 Table of Contents ...................................................................................................................................... 3
   0.5 Terminology and Document Conventions ............................................................................................. 4
   0.6 Revision History ...................................................................................................................................... 4

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

2 Application Domain ........................................................................................................................................... 5
   2.1 Asynchronous programming models ........................................................................................................ 6
   2.2 Mixture of programming models ............................................................................................................. 6
   2.3 Terminology + Abbreviations .................................................................................................................. 6

3 Problem Description ......................................................................................................................................... 7
   3.1 Asynchronous Services .......................................................................................................................... 7

4 Requirements ...................................................................................................................................................... 7

5 Technical Solution ............................................................................................................................................... 8
   5.1 The Async Service .................................................................................................................................. 8
   5.2 Async Proxies ......................................................................................................................................... 8
       5.2.1 Class Proxying .................................................................................................................................. 9
       5.2.2 Async Proxy return types ................................................................................................................ 9
0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

0.6 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>08/11/13</td>
<td>Tim Ward – Initial version of RFC</td>
</tr>
<tr>
<td>0.2</td>
<td>15/11/13</td>
<td>Virtual F2F comments – update numbering of requirements to match Final RFP version; use DS in example code to demonstrate good practice.</td>
</tr>
</tbody>
</table>
1 Introduction

OSGi Bundles collaborate using loosely coupled services registered in the OSGi service registry. This is a powerful and flexible model, and allows for the dynamic replacement of services at runtime. OSGi services are therefore a very common interaction pattern within OSGi.

As with most Java APIs and Objects, OSGi services are primarily synchronous in operation. This has several benefits; synchronous APIs are typically easier to write and to use than asynchronous ones; synchronous APIs provide immediate feedback; synchronous implementations typically have a less complex threading model.

Asynchronous APIs, however, have different advantages. Asynchronous APIs can reduce bottlenecks by encouraging more effective use of parallelism, improving the responsiveness of the application. This intent of this RFC is to allow clients to get the benefits of asynchronous invocation, even when the Service API has been written in a synchronous way.
2 Application Domain

This section explores various aspects of adding support for asynchronous execution. Asynchronous execution typically is achieved via the introduction of a queuing mechanism for “tasks” which are pulled in and executed by one or more Threads. In the case of remote invocations, the task queue is often on the remote machine, allowing the request to be sent and executed without occupying a local Thread. These mechanisms are often also used to handle events, for example the OSGi Event Admin Service provides an asynchronous communication model.

Synchronous invocations are typically easier to program, but once a client makes a request, either local or remote, the client is blocked waiting for execution to complete and return control to the client. While asynchronous execution may be more complex to program, it offers many benefits and advantages.

For example, synchronous remote invocations depend on the availability of the network during request execution. If a client or server fails during the execution of a request, the request typically has to be resubmitted. This may not be a problem for some applications, where it’s easy to re-create the request input. But for other applications, such as an ATM, gas pump, or electronic funds transfer, it may not be easy to recapture the input data and create another request message, and asynchronous protocols meet the requirement better. Even when it is possible to recreate a request message, it is not always easy to know at which point the server failed – i.e. whether or not an update was performed as a result of executing the request, and if so, whether performing the update a second time might cause data inconsistency. And in this case asynchronous protocols can also offer some advantages.

Synchronous invocations operate on a first-come, first-served scheduling mechanism (i.e. the computer has to process requests as they are made by the caller). This means that it's not easy to treat some invocations with higher priority than others, although this is a common application requirement (for example, a bank wants to process the outstanding $1M deposits ahead of the $10 deposits near the end of the banking day). As they have work queues, asynchronous processing engines can process work in an arbitrary order if they choose.

2.1 Asynchronous programming models

A variety of asynchronous programming models and frameworks are successfully used in enterprise applications today, including ExecutorServices, Async EJBs, Async Servlets, Node.js, store-and-forward, pub-sub, and broadcast/multicast to name a few. These programming models assume that a task is visible to a program using one or more asynchronous submission mechanisms (for example, JMS) and that the program is responsible for explicitly creating or retrieving a response using the API and then may act upon it in a way that is visible to another program using the same API.

For example, a store and forward system has one program submitting a message to a queue using a SEND or SUBMIT command, and another program retrieving the message from a queue using a RECEIVE or DEQUEUE command from the asynchronous programming model API. The sending program is responsible for packing, or serializing, the message, and the receiving program is responsible for unpacking, or de-serializing the message. (Some APIs define a wire format while others do not.)

In each case, management utilities are required to configure the capabilities of the asynchronous implementation being used so that they are able to reject work when overloaded, make best use of the resources available, and to identify, report and resolve any errors that may occur.
2.2 Mixture of programming models

Many enterprise applications require both synchronous and asynchronous execution models for different types of IT functions. For a reserved ticket purchase, for example, it may be necessary to synchronize the database update with the reply to the user to indicate the ticket was purchased, since only one person can have a given seat. For a book purchase, however, it may be sufficient to reply to the user that the order was received, and that it would be fulfilled later. Some of the fulfillment operations for a book order might also use synchronous communications, for example to debit inventory while packing the order for shipment.

2.3 Terminology + Abbreviations

Client – Application code that wishes to call one or more OSGi services asynchronous

Target Service – A service that is to be called asynchronously by the client

Async Service – The OSGi service representing the Asynchronous Services implementation. Used by the client.

Async Mediator – A mediator object representing the target service, created by the Async Service

Success Callback – A callback made when an asynchronous invocation exits with a normal return value

Failure Callback – A callback made when an asynchronous invocation exits by throwing an exception

Completion Callback – A callback that is made when an asynchronous invocation exits, regardless of how it exits.

Asynchronous Invocation – A single method call that is to be executed without blocking the requesting thread.

Asynchronous Task – An aggregate of one or more asynchronous invocations. The invocations that make up a task may run in parallel, or sequentially, or a mixture of both.

3 Problem Description

The current OSGi programming model for communications among components and bundles is based on the OSGi service interface, which implies a synchronous semantic (i.e. the client invokes on the interface and waits for the reply), and language objects as parameters. These characteristics are typical of local invocations and distributed RPC and meet many requirements, but we want to extend these capabilities to support asynchronous invocation.

3.1 Asynchronous Services

We propose that the EEG evaluate options for specifying Asynchronous invocation of services – specifically the ability for a client to issue an invocation on a service interface without waiting for completion, and relying on a later notification or polling to check completion and retrieve results. For illustration, a low-level equivalent of such a framework is provided in J2SE by the Future interface. Other technologies (such as CORBA) provide asynchronous 'one-way' support on their remote interfaces. There are significant design considerations involved.
in selecting whether this may be defined within the “OSGi Services” architecture, and/or “Blueprint”, and/or Remote Services; and how a particular choice of solution relates to all three architectures.

4 Requirements

AS01 – The solution MUST provide a standard client-side API for making asynchronous invocations on existing, synchronous, OSGi services, where the invocation returns quickly and a return value can be obtained later.

AS02 – The solution MUST allow transparent delegation to services that are already implemented in an asynchronous fashion, therefore servicing the asynchronous requests through their own implementations.

AS03 – The solution MAY provide a synchronous client-side API to services which are implemented in an asynchronous fashion.

AS04 – The solution MUST allow for one-way (fire and forget) asynchronous services.

AS05 – The solution MUST support Promises, where invocations can be made that later return a value.

AS06 – The solution SHOULD support callbacks when asynchronous executions complete, both successfully and unsuccessfully.

AS07 – The solution MUST be applicable to both local OSGi Services as well as Remote OSGi Services.

AS08 – The solution MUST be fully backwards compatible with existing OSGi Service and Service Registry usage.

AS09 – The solution SHOULD be sympathetic to Java 8’s lambda support, meaning callbacks should follow the Single Abstract Method principle where possible.

AS10 – The solution MUST define a mechanism that allows service providers to advertise an asynchronous mode of operation if they support it.

5 Technical Solution

In order for a client to make asynchronous invocations on a service there are several necessary steps. First it is necessary to identify the service to be invoked, which we shall refer to as the target service. In the absence of any
further support the client would then need to create a Runnable or Callable that invoked the target service, and then execute this using an Executor or by starting a new Thread.

Rather than having each client managing its own asynchronous Executor an Async Service can manage the execution of the asynchronous invocation. This requires the Async service to track the invocations made on the target service and to asynchronously service them. To support requirements AS05 and AS06 the Async service also needs to provide a mechanism to register callbacks, and to return a Promise.

This requirement is similar to the requirements that mocking frameworks such as Mockito have. They track invocations on proxy objects so that the invocations can later be checked, or so that specific invocations can be configured to return particular values. The Async Service uses a similar pattern, where invocations on a mediator are used to register the asynchronous executions that should occur.

5.1 OSGi Promises

One of the fundamental pieces of an asynchronous programming model is the mechanism by which clients retrieve the result of the asynchronous work. Since Java 5 there has been a java.util.concurrent.Future interface available in the core Java runtime, which means that it is the de-facto Promise API in Java. Futures have some limitations however, in that they have no mechanism for registering callbacks. This shortcoming will be addressed in Java 8 with the introduction of java.util.concurrent.CompletableFuture, however this is also unsuitable for use in an OSGi specification. OSGi therefore requires its own Promise API to support Asynchronous Services. As the API is generally applicable to asynchronous programming it is anticipated that it may be used more widely. As such it should be able to be used independently of Asynchronous Services. It would also be advantageous for OSGi Promises to be usable outside of an OSGi framework, therefore the Promises API should avoid having any dependency on other OSGi APIs, such as the core framework package. In addition, although the Promise interface itself will not extend java.util.concurrent.Future the specification should ensure that implementation classes are not precluded from also implementing Future. This means that method names on the Promise interface should not clash with those on the Future interface unless they have exactly the same semantics.

5.1.1 The Promise

The primary interface for OSGi Promises is the org.osgi.util.promise.Promise<T>. There are several important things that a Promise must be able to achieve:

1. It must be possible to determine whether an asynchronous execution has completed, and to get the result of the asynchronous execution, and to retrieve the failure if the execution completed exceptionally. This can be achieved using the isDone() getValue() and getError() methods:

   • public boolean isDone(); is used to determine whether a Promise has resolved. If the promise has resolved then this method returns true, otherwise it returns false. This method should not block, If the method returns true then the Promise should not block in subsequent calls to get() and getError()

   • public T getValue(); is used to retrieve the result of the asynchronous task. It blocks until the promise is resolved, or until the calling thread is interrupted. If the thread is interrupted then InterruptedException is thrown. If the promise resolves successfully then the result is returned. If the promise resolves with a failure then the failure is wrapped in an InvocationTargetException and thrown to the client.

   • public Throwable getError(); is used to retrieve the result of a failed execution. It blocks until the promise is resolved, or until the calling thread is interrupted. If the thread is interrupted then InterruptedException is thrown. If the promise resolves successfully then null is returned. If the promise resolves with a failure then the exception that was thrown is returned to the client.
2. It must be possible to register callbacks with the promise that are called after the promise is resolved. These callbacks need to be able to access the result of the execution. This is achieved through the use of the two `then(...)` methods and `onresolve()`:

- `public <R, S extends R> Promise<R> then(Success<? super T, R> success)` and `public <R, S extends R> Promise<R> then(Success<? super T, R> success, Failure failure)` are the mechanism by which clients can register callbacks. Success callbacks are called after a promise has resolved with a value, Failure callbacks are called after a promise with a failure. In both cases there is a “happens before” relationship, which means that the promise will always return true for calls to `isDone()` and will not block when retrieving a value. The thread that is used to execute the callbacks is undefined and implementation dependent. If there are multiple callbacks defined for a single promise then they may be called concurrently on separate threads, or sequentially on a single thread. Client calls to `then(...)` return another Promise. This is known as “chaining”, and the chained promise resolves in one of four ways:

1. The parent promise resolves with a failure. In this case the chained promise resolves immediately with the same failure.

2. The parent promise resolves successfully, but the Success callback throws an exception. In this case the chained promise resolves immediately with the exception thrown by the callback.

3. The parent promise resolves successfully, and the Success callback is `null`, or returns `null`. In this case the chained promise resolves immediately with a successful value of `null`

4. The parent promise resolves successfully, and the Success callback returns a Promise. In this case the chained promise resolves when the returned promise resolves. The resolution of both promises must be the same value.

- `public void onresolve(Runnable toCall);` registers a runnable to be called when the promise resolves, either successfully or unsuccessfully. This has the same “happens before” behaviour as the callbacks from the `then()` methods, and also runs on an undefined thread. It does not make sense for the client to pass `null` to this method. If this situation occurs then the Promise should throw a `NullPointerException`. If an exception is thrown by the run() method of the callback then this must not prevent the processing of other callbacks associated with the promise.

5.1.2 The Deferred class

Promises may be implemented in many different ways, however there is significant value in having a simple implementation available as part of the promise API. The `org.osgi.util.promise.Deferred` class provides a simple implementation that allows implementations to easily provide basic promise behaviour.

The `Deferred` object is instantiated by the code that wishes to perform the asynchronous work. It then uses the `getPromise()` method to obtain a promise to return to the client. When the asynchronous work has completed the asynchronous code then calls either the `resolve()` method or the `fail()` method on the deferred object. If the resolution of the Deferred directly corresponds to the resolution of another promise then this can be simplified by using the `resolveWith(Promise p)` method.
5.1.3 The Promises utility class

There are some common usage patterns involving promises that can benefit from helper utility methods. These methods are defined as static methods of the `org.osgi.util.promise.Promises` class.

Latch promises

One common use case is that a client wishes to be notified when a group of promises are all complete. This can be achieved using the `Promise<Void> newLatchPromise(Promise<?>... promises)` or `Promise<Void> newLatchPromise(Collection<Promise<T>> promises)` methods. These methods take a number of promises, and return a promise that resolves once all of the promises passed to the method have resolved. If any of the promises fail then the aggregate promise will eventually resolve with a `FailedPromisesException`. This exception provides access to the failed promises.

Direct promises

In some cases it can be useful to create a promise representing a value that has already been calculated (for example in a Success callback). The Promises class offers two static methods `newResolvedPromise(T object)`, and, `newFailedPromise(Throwable object)` to simplify this case.

5.1.4 Functional Programming with Promises

Asynchronous program designs often make use of functional programming concepts to simplify their implementation. Another use-case for the Promise API is therefore to provide basic functional programming features from which more complex behaviours can be composed.

Functional interface types

Java 8 is introducing a new package, `java.util.function`, which contains a number of interfaces used to support functional-style programming within Java. The two most widely used and useful of these interfaces are `Predicate<T>`, which can be used to provide a simple true or false test, and `Function<T,R>` which takes an argument and returns a value. As OSGi is not currently able to depend upon Java 8 it is necessary for OSGi to provide equivalent interfaces for use with Promises. This RFC proposes that these interfaces are provided in the `org.osgi.util.function` package. At some point in the future these interfaces may be updated to extend the core types from Java 8, and should therefore have exactly the same method signatures.

Functional behaviours for promises

To simplify its usage in functional-style programs, the Promise interface declares the following methods:

- `<R, S extends R> Promise<R> map(Function<? super T, S> f)` – When the promise resolves successfully then the result is transformed using the function. If the function throws an exception then that exception is used to fail the returned promise. If the original promise fails then the returned promise is failed with the same failure.

- `Promise<T> filter(Predicate<? super T> p)` – When the promise resolves successfully then the result is tested using the predicate. If the predicate returns true then the returned promise is resolved with the same value as the original promise. If the predicate returns false then the returned promise is failed with a NoSuchElementException. If the original promise is resolved with a failure then the returned promise is resolved with the same failure.

- `<R, S extends R> Promise<R> flatMap(Function<? super T, Promise<S>> f)` – When the promise resolves successfully then the result is passed to the supplied function. The promise returned
by this function is then used to resolve or fail the returned promise as appropriate. If the function throws an exception then that exception is used to fail the returned promise. If the original promise fails then the returned promise is failed with the same failure.

• \(<S \text{ extends } T> \text{ Promise}\langle T \rangle \text{ recover}(\text{Function}\langle \text{Promise}\langle ?\rangle, S \rangle \text{ f}) – When the promise resolves successfully then the returned promise is resolved with the same value. If the promise fails then it is passed to the supplied function. The value returned by this function is then used to resolve the returned promise. If the function throws an exception then that exception is used to fail the returned promise. If the function is null then the returned promise will fail with the same failure as the original promise.

• \(<S \text{ extends } T> \text{ Promise}\langle T \rangle \text{ recoverWith}(\text{Function}\langle \text{Promise}\langle ?\rangle, \text{ Promise}\langle S\rangle \rangle \text{ f}) – When the promise resolves successfully then the returned promise is resolved with the same value. If the promise fails then it is passed to the supplied function. The promise returned by this function is then used to resolve or fail the returned promise as appropriate. If the function throws an exception then that exception is used to fail the returned promise. If the function is null then the returned promise will fail with the same failure as the original promise.

• \(<S \text{ extends } T> \text{ Promise}\langle T \rangle \text{ fallbackTo}(\text{Promise}\langle S \rangle \text{ f}) – When the promise resolves successfully then the returned promise is resolved with the same value. If the promise fails and the supplied promise resolves successfully then the returned promise is resolved with the value from the supplied promise. If the supplied promise also fails then the returned promise will be failed with the same failure as the original promise.

5.2 The Async Service

The Async service is the primary interaction point between a client and the Async Services implementation. An Async Services implementation must expose a service implementing the org.osgi.service.async.Async interface.

Clients obtain an instance of the Async Service using the normal OSGi service registry mechanism, either directly using the OSGi framework API, or using dependency injection.

Implementations of the Async service must be thread safe. They should be safe to use simultaneously across multiple clients and from multiple threads within the same client.

5.3 Async Mediators

When a client has chosen a target service, it can use the Async service to make an asynchronous invocation. The first step is to use the Async Service to create a mediator for the real service.

\(<T> T \text{ mediate}(\text{ServiceReference}\langle T \rangle \text{ ref});\)

Example:

```java
private Async asyncService;
private ServiceReference<List> listRef;
@Reference
synchronized void setAsync(Async async) {
    asyncService = async;
}
@Reference(service = List.class)
synchronized void setList(ServiceReference<List> list) {
```
5.3.1 Generating a Mediator

When creating the Async Mediator object the Async Service should attempt to load all of the classes listed in the objectClass property of the service reference using the client bundle. Any ClassNotFoundException thrown when attempting to load these classes should be ignored. The loaded classes should then be divided into Java interfaces and concrete classes.

The async service must then generate a mediator object. If the service only advertises Java interfaces in its objectClass property then the mediator object must implement all of the Java interfaces that could be loaded by the client bundle. The mediator class must be defined using the client bundle's classloader. This can easily be achieved using the java.lang.reflect.Proxy class.

5.3.2 Generating a Mediator for Concrete Classes

If a service is registered advertising one or more concrete class types then generating a mediator requires more complex handling. In this case the mediator object created by the Async service should also inherit from the most specialised concrete type listed in the target service’s objectclass property that can be loaded using the client bundle's ClassLoader. There are three reasons why the Async service may not be able to mediate a class type:

1. The most specialised type is final
2. The most specialised type has no zero-argument constructor
3. One or more public methods present in the type hierarchy (other than those declared by java.lang.Object) are final.

If any of these rules are violated then the Async service should fall back to creating an interface-only mediator.

5.3.3 Async Mediator return types

When invoked the Async mediator should return rapidly (i.e. it should not perform blocking operations). The client should not attempt to interpret the returned value. The value may be null (or null-like in the case of primitives) or contain implementation specific information.

5.3.4 Thread safety

Whilst the Async Service itself must be thread safe, async mediator objects may not be. Clients should avoid sharing async mediator objects between threads if they wish to be portable between implementations.

5.3.5 Mediating other objects

The Async service also allows mediators to be created for arbitrary objects (rather than for Service References) using the method:

```
<T> T mediate(T object);
```

In this case the mediator generation behaviour is the same as mediating a service reference which lists the entire Java class and interface hierarchy for the supplied object.
5.4 Building simple tasks

Once a mediator has been created it can be used, in conjunction with the Async service, to run an asynchronous task.

5.4.1 Running an asynchronous task

To begin an asynchronous task the client invokes a method on the asynchronous mediator. The client then passes the result of that invocation to the Async service to begin the asynchronous invocation. These two calls must occur on the same thread, in particular it is an error for a client to invoke the mediator with one thread and then to attempt to begin that asynchronous task with a different thread. The execution has an associated type, which is the return type of the asynchronous invocation (or the associated wrapper type for primitives and void).

Example:

```java
private Async asyncService;
private ServiceReference<List> listRef;

@Reference
synchronized void setAsync(Async async) {
    asyncService = async;
}

@Reference(service = List.class)
synchronized void setList(ServiceReference<List> list) {
    listRef = list;
}

public synchronized void doStuff() {
    List mediator = asyncService.mediate(ref);
    Promise<Boolean> promise = asyncService
        .call(mediator.contains("badEntry"));
    ...
}
```

At this point the Asynchronous task is running, and may have already completed. It can be interacted with using the Promise returned by the call method.

Once a context has been established, clients can continue to start other asynchronous tasks, or to chain Promises with the one returned by the async service.

5.4.2 Establishing context for Void methods

In Java void methods have no return value, and therefore cannot return anything. This means that void methods cannot establish context in the same way that other methods do. Void methods therefore need to be declared in a different way, the client calls the void method as a separate statement and then either:

- passes null to the async service.

  or

- calls the no-args call() method

Example – Out of Line expression:
private Async asyncService;
private ServiceReference<List> listRef;

@Reference
synchronized void setAsync(Async async) {
    asyncService = async;
}

@Reference(service = List.class)
synchronized void setList(ServiceReference<List> list) {
    listRef = list;
}

public synchronized void doStuff() {
    List mediator = asyncService.createAsyncMediator(ref);
    mediator.clear();
    asyncService.call();
}

Promises for void methods are resolved with null when they are successful.

5.4.3 Detecting API usage errors

Async service implementations should make a best-effort attempt to validate the usage of the API, and may throw 
IllegalStateException if an API usage error is detected. Implementations must throw an 
IllegalStateException in 
the case where the client attempts to start an asynchronous invocation without first invoking a method on the 
mediator object. Implementations should also throw 
IllegalStateException if they are able to detect that there have 
been multiple calls to the mediator without starting an asynchronous invocation.

5.4.4 Fire-and-forget tasks

Sometimes a client does not require notification that an asynchronous invocation has completed. In this case the 
client could use one of the existing call() methods and simply discard the returned Promise object. This, 
however, can be wasteful of resources. The act of resolving the Promise object may be expensive, for example it 
may involve serializing a return value over a network.

If the client can communicate to the async services implementation that the Promise object is not needed then the 
async service can potentially optimise the asynchronous invocation. To indicate that the client wants to make a 
fire-and-forget style call the client invokes the mediator as normal, but then begins the asynchronous invocation 
using the execute() method.

Example:

```java
private Async asyncService;
private ServiceReference<List> listRef;

@Reference
synchronized void setAsync(Async async) {
    asyncService = async;
}

@Reference(service = List.class)
synchronized void setList(ServiceReference<List> list) {
    listRef = list;
}

public synchronized void doStuff() {
    List mediator = asyncService.createAsyncMediator(ref);
    mediator.clear();
    asyncService.call();
}
```
When the Async service receives a call to the `execute()` method it should make the same API usage checks as for the `call()` methods. It should also start the asynchronous invocation, however it does not need to create a Promise representing the state of the invocation.

### 5.5 Building more complex tasks

Asynchronous tasks may consist of multiple distinct asynchronous invocations. The promise API allows tasks to be chained, or to wait for a set of tasks to have completed.

### 5.6 Execution Failures

There are a variety of reasons that Asynchronous invocations may fail. In any of these cases the asynchronous invocation should resolve with an `org.osgi.framework.ServiceException`, with a type of `ASYNC` (see OSGi core R6). This exception should be passed to any failure callbacks as normal:

- If the client bundle's bundle context becomes invalid before looking up the target service. This does not apply if the mediated object was not created with a ServiceReference.
- If the target service becomes unavailable before making the asynchronous invocation, or returns null on lookup. This does not apply if the mediated object was not created with a ServiceReference.
- If the Async service is unable to accept new work, for example it is in the process of being shut down.
- If the target service is unable to be invoked with the supplied arguments (this indicates a missing uses constraint).

If the target service is successfully invoked, but the method call throws an exception, then this should be used to resolve the promise without wrapping it.

### 5.7 Delegating to asynchronous implementations

Some service APIs are already asynchronous in operation, and others are partly asynchronous, in that some methods run asynchronously and others do not. There are also services which have a synchronous API, but could run asynchronously because they are a proxy to another service. A good example of this kind of service is a remote service. Remote services are local views of a remote endpoint, and depending upon the implementation of the endpoint it may be possible to make the remote call asynchronously, optimizing the thread usage of any local asynchronous call.

Services that already have some level of asynchronous support can advertise this by implementing the `org.osgi.service.async.spi.AsyncDelegate` interface. This can be used by the asynchronous services implementation, or by the client directly, to indicate that a call made on the service should be processed asynchronously.

#### 5.7.1 Calls that return a Promise

For calls that return a promise, the `AsyncDelegate` can be used as follows:

1. Cast the object to `AsyncDelegate`
2. Invoke the `async(Method, Object[])` method, holding on to the `Promise` returned by the invocation
   - If the returned promise is null then this service does not support asynchronous invocation of that method, and should be treated as a normal synchronous service.
   - If the async call throws an exception then the promise should immediately be resolved with that exception.

3. The `AsyncDelegate` will begin asynchronously executing the method, when the returned promise resolves it should be used to resolve the promise returned to the client by the async service.

### 5.7.2 Fire and forget calls

Fire and forget calls do not need to return a promise, and may therefore be more aggressively optimised by the `AsyncDelegate`. These can be handled as follows:

1. **Cast the object to `AsyncDelegate`**
2. **Invoke the `execute(Method, Object[])` method**
   - If the execute method returns false then this service does not support asynchronous invocation of that method, and should be treated as a normal synchronous service.
   - If the execute method throws an exception then the async service should log this exception. The exception should not be thrown to the client
3. The `AsyncDelegate` will begin asynchronously executing the method, and need have no further interaction with the async service.

### 6 Data Transfer Objects

It is unclear whether Asynchronous Services would benefit from DTOs
7 Javadoc

7.1 The Promises API
<table>
<thead>
<tr>
<th>Package Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.util.function</td>
<td>20</td>
</tr>
<tr>
<td>org.osgi.util.promise</td>
<td>23</td>
</tr>
</tbody>
</table>

Function Package Version 1.0.
Promise Package Version 1.0.
Package org.osgi.util.function

@org.osgi.annotation.versioning.Version(value="1.0")

Function 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>Function</td>
<td>A function that accepts a single argument and produces a result.</td>
<td>21</td>
</tr>
<tr>
<td>Predicate</td>
<td>A predicate that accepts a single argument and produces a boolean result.</td>
<td>22</td>
</tr>
</tbody>
</table>

Package org.osgi.util.function Description

Function Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle’s manifest.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.util.function; version="(1.0,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.util.function; version="(1.0,1.1)"
```
### Interface Function

**org.osgi.util.function**

**Type Parameters:**
- `T` - The type of the function input.
- `R` - The type of the function output.

@org.osgi.annotation.versioning.Consumertype

```java
public interface Function
```

A function that accepts a single argument and produces a result.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

**ThreadSafe**

---

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply(T t)</td>
<td>21</td>
</tr>
</tbody>
</table>

Applies this function to the specified argument.

---

### Method Detail

**apply**

```java
R apply(T t)
```

Applies this function to the specified argument.

**Parameters:**
- `t` - The input to this function.

**Returns:**
- The output of this function.
Interface Predicate

Type Parameters:

T - The type of the predicate input.

@org.osgi.annotation.versioning.ConsumerType
public interface Predicate

A predicate that accepts a single argument and produces a boolean result.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

ThreadSafe

Method Summary

<table>
<thead>
<tr>
<th>boolean test(T t)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluates this predicate on the specified argument.</td>
<td>22</td>
</tr>
</tbody>
</table>

Method Detail

test

boolean test(T t)

Evaluates this predicate on the specified argument.

Parameters:

- t - The input to this predicate.

Returns:

true if the specified argument is accepted by this predicate; false otherwise.
## Package org.osgi.util.promise

@org.osgi.annotation.versioning.Version(value="1.0")

Promise Package Version 1.0.

See: [Description](#)

### Interface Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Failure</strong></td>
<td>Failure callback for a Promise.</td>
</tr>
<tr>
<td><strong>Promise</strong></td>
<td>A Promise of a value.</td>
</tr>
<tr>
<td><strong>Success</strong></td>
<td>Success callback for a Promise.</td>
</tr>
</tbody>
</table>

### Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deferred</strong></td>
<td>A Deferred Promise resolution.</td>
</tr>
<tr>
<td><strong>Promises</strong></td>
<td>Static helper methods for <a href="#">Promises</a>.</td>
</tr>
</tbody>
</table>

### Exception Summary

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FailedPromisesException</strong></td>
<td>Promise failure exception for a collection of failed Promises.</td>
</tr>
</tbody>
</table>

## Package org.osgi.util.promise Description

Promise Package Version 1.0.

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

Example import for consumers using the API in this package:

Import-Package: org.osgi.util.promise; version="[1.0,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.util.promise; version="[1.0,1.1)"
Class Deferred

java.lang.Object

org.osgi.util.promise

Type Parameters:

T - The value type associated with the created Promise.

public class Deferred
extends Object

A Deferred Promise resolution.

Instances of this class can be used to create a Promise that can be resolved in the future. The associated Promise can be successfully resolved with `resolve(Object)` or resolved with a failure with `fail(Throwable)`. It can also be resolved with the resolution of another promise using `resolveWith(Promise)`. The associated Promise can be provided to any one, but the Deferred object should be made available only to the party that will responsible for resolving the Promise.

Immutable

Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deferred()</td>
<td>Create a new Deferred with an associated Promise.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fail(Throwable failure)</td>
<td>Fail the Promise associated with this Deferred.</td>
</tr>
<tr>
<td>getPromise()</td>
<td>Returns the Promise associated with this Deferred.</td>
</tr>
<tr>
<td>resolve(T value)</td>
<td>Successfully resolve the Promise associated with this Deferred.</td>
</tr>
<tr>
<td>resolveWith(Promise&lt;? extends T&gt; with)</td>
<td>Resolve the Promise associated with this Deferred with the specified Promise.</td>
</tr>
</tbody>
</table>

Constructor Detail

Deferred

public Deferred()

Create a new Deferred with an associated Promise.

Method Detail

getPromise

public Promise<T> getPromise()

Returns the Promise associated with this Deferred.
Returns:
The Promise associated with this Deferred.

public void resolve(T value)

Successfully resolve the Promise associated with this Deferred.

After the associated Promise is resolved with the specified value, all registered callbacks are called and any chained Promises are resolved.

Resolving the associated Promise happens-before any registered callback is called. That is, in a registered callback, Promise.isDone() must return true and Promise.getValue() and Promise.getFailure() must not block.

Parameters:
value - The value of the resolved Promise.

Throws:
IllegalStateException - If the associated Promise was already resolved.

public void fail(Throwable failure)

Fail the Promise associated with this Deferred.

After the associated Promise is resolved with the specified failure, all registered callbacks are called and any chained Promises are resolved.

Resolving the associated Promise happens-before any registered callback is called. That is, in a registered callback, Promise.isDone() must return true and Promise.getValue() and Promise.getFailure() must not block.

Parameters:
failure - The failure of the resolved Promise. Must not be null.

Throws:
IllegalStateException - If the associated Promise was already resolved.

public Promise<Void> resolveWith(Promise<? extends T> with)

Resolve the Promise associated with this Deferred with the specified Promise.

If the specified Promise is successfully resolved, the associated Promise is resolved with the value of the specified Promise. If the specified Promise is resolved with a failure, the associated Promise is resolved with the failure of the specified Promise.

After the associated Promise is resolved with the specified Promise, all registered callbacks are called and any chained Promises are resolved.

Resolving the associated Promise happens-before any registered callback is called. That is, in a registered callback, Promise.isDone() must return true and Promise.getValue() and Promise.getFailure() must not block.

Parameters:
with - A Promise whose value or failure will be used to resolve the associated Promise. Must not be null.
Returns:
A Promise that is resolved only when the associated Promise is resolved by the specified Promise. The returned Promise will be successfully resolved, with the value null, if the associated Promise was resolved by the specified Promise. The returned Promise will be resolved with a failure of IllegalStateException if the associated Promise was already resolved when the specified Promise was resolved.
Class FailedPromisesException

org.osgi.util.promise

java.lang.Object
  ▼ java.lang.Throwable
    ▼ java.lang.Exception
      ▼ java.lang.RuntimeException
        ▼ org.osgi.util.promise.FailedPromisesException

All Implemented Interfaces:
  Serializable

public class FailedPromisesException
extends RuntimeException

Promise failure exception for a collection of failed Promises.

Constructor Summary

<table>
<thead>
<tr>
<th>FailedPromisesException</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Collection&lt;Promise&lt;?&gt;&gt; failed)</td>
<td>27</td>
</tr>
</tbody>
</table>

Create a new FailedPromisesException with the specified Promises.

Parameters:
  failed - A collection of Promises that have been resolved with a failure. Must not be null.

Method Summary

<table>
<thead>
<tr>
<th>Collection&lt;Promise&lt;?&gt;&gt; getFailedPromises()</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the collection of Promises that have been resolved with a failure.</td>
<td>27</td>
</tr>
</tbody>
</table>

Constructor Detail

FailedPromisesException

public FailedPromisesException(Collection<Promise<?>> failed)

Create a new FailedPromisesException with the specified Promises.

Parameters:
  failed - A collection of Promises that have been resolved with a failure. Must not be null.

Method Detail

getFailedPromises

public Collection<Promise<?>> getFailedPromises()

Returns the collection of Promises that have been resolved with a failure.

Returns:
  The collection of Promises that have been resolved with a failure. The returned collection is unmodifiable.
Interface Failure

org.osgi.util.promise

@org.osgi.annotation.versioning.ConsumerType
default interface Failure

Failure callback for a Promise.

A Failure callback is registered with a Promise using the Promise.then(Success, Failure) method and is called if the Promise is resolved with a failure.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

ThreadSafe

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>void fail(Promise&lt;?&gt; resolved)</td>
<td>28</td>
</tr>
</tbody>
</table>

Failure callback for a Promise.

This method is called if the Promise with which it is registered resolves with a failure.

In the remainder of this description we will refer to the Promise returned by Promise.then(Success, Failure) when this Failure callback was registered as the chained Promise.

If this method completes normally, the chained Promise will be failed with the same exception which failed the resolved Promise. If this method throws an exception, the chained Promise will be failed with the thrown exception.

Parameters:
resolved - The failed resolved Promise.

Throws:
Exception - The chained Promise will be failed with the thrown exception.
Interface Promise

`org.osgi.util.promise`

**Type Parameters:**

T - The value type associated with this Promise.

```java
@org.osgi.annotation.versioning.ProviderType
public interface Promise
```

A Promise of a value.

A Promise represents a future value. It handles the interactions to for asynchronous processing. A `Deferred` object can be used to create a Promise and later resolve the Promise. A Promise is used by the caller of an asynchronous function to get the result or handle the error. The caller can either get a callback when the Promise is resolved with a value or an error, or the Promise can be used in chaining. In chaining, callbacks are provided that receive the resolved Promise, and a new Promise is generated that resolves based upon the result of a callback.

Both [callbacks](#) and [chaining](#) can be repeated any number of times, even after the Promise has been resolved.

**Example callback usage:**

```java
final Promise<String> foo = foo();
foo.onResolve(new Runnable() {
    public void run() {
        System.out.println(foo.getValue());
    }
});
```

**Example chaining usage:**

```java
Success<String,String> doubler = new Success<String,String>() {
    public Promise<String> call(Promise<String> p) throws Exception {
        return Promises.newResolvedPromise(p.getValue()+p.getValue());
    }
};
final Promise<String> foo = foo().then(doubler).then(doubler);
foo.onResolve(new Runnable() {
    public void run() {
        System.out.println(foo.getValue());
    }
});
```

**ThreadSafe**

### Method Summary

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Promise&lt;T&gt;</code></td>
<td>Page</td>
</tr>
<tr>
<td><code>fallbackTo(Promise&lt;? extends T&gt; fallback)</code></td>
<td>42</td>
</tr>
<tr>
<td>Fall back to the value of the specified Promise if this Promise fails.</td>
<td></td>
</tr>
<tr>
<td><code>Promise&lt;T&gt;</code></td>
<td>33</td>
</tr>
<tr>
<td><code>filter(Predicate&lt;? super T&gt; predicate)</code></td>
<td>Filter the value of this Promise.</td>
</tr>
<tr>
<td><code>Promise&lt;R&gt;</code></td>
<td>34</td>
</tr>
<tr>
<td><code>flatMap(Function&lt;? super T,R&gt; mapper)</code></td>
<td>FlatMap the value of this Promise.</td>
</tr>
<tr>
<td><code>Promise&lt;R&gt;</code></td>
<td>30</td>
</tr>
<tr>
<td><code>getFailure()</code></td>
<td>Returns the failure of this Promise.</td>
</tr>
<tr>
<td><code>T</code></td>
<td>30</td>
</tr>
<tr>
<td><code>getValue()</code></td>
<td>Returns the value of this Promise.</td>
</tr>
<tr>
<td><code>boolean</code></td>
<td>30</td>
</tr>
<tr>
<td><code>isDone()</code></td>
<td>Returns whether this Promise has been resolved.</td>
</tr>
<tr>
<td><code>Promise&lt;R&gt;</code></td>
<td>33</td>
</tr>
<tr>
<td><code>map(Function&lt;? super T,R&gt; mapper)</code></td>
<td>Map the value of this Promise.</td>
</tr>
</tbody>
</table>
**Method Detail**

### isDone

```java
boolean isDone()
```

Returns whether this Promise has been resolved.

This Promise may be successfully resolved or resolved with a failure.

**Returns:**
true if this Promise was resolved either successfully or with a failure; false if this Promise is unresolved.

### getValue

```java
T getValue()
```

Throws: `InvocationTargetException`, `InterruptedException`

Returns the value of this Promise.

If this Promise is not resolved, this method must block and wait for this Promise to be resolved before completing.

If this Promise was successfully resolved, this method returns with the value of this Promise. If this Promise was resolved with a failure, this method must throw an `InvocationTargetException` with the `failure exception` as the cause.

**Returns:** The value of this resolved Promise.

**Throws:**
- `InvocationTargetException` - If this Promise was resolved with a failure. The cause of the `InvocationTargetException` is the failure exception.
- `InterruptedException` - If the current thread was interrupted while waiting.

### getFailure

```java
Throwable getFailure()
```

Throws `InterruptedException`

Returns the failure of this Promise.

If this Promise is not resolved, this method must block and wait for this Promise to be resolved before completing.
If this Promise was resolved with a failure, this method returns with the failure of this Promise. If this Promise was successfully resolved, this method must return null.

**Returns:**
The failure of this resolved Promise or null if this Promise was successfully resolved.

**Throws:**
InterruptedException - If the current thread was interrupted while waiting.

---

### onResolve

```java
Promise<T> onResolve(Runnable callback)
```

Register a callback to be called when this Promise is resolved.

The specified callback is called when this Promise is resolved either successfully or with a failure.

This method may be called at any time including before and after this Promise has been resolved.

Resolving this Promise happens-before any registered callback is called. That is, in a registered callback, `isDone()` must return true and `getValue()` and `getFailure()` must not block.

A callback may be called on a different thread than the thread which registered the callback. So the callback must be thread safe but can rely upon that the registration of the callback happens-before the registered callback is called.

**Parameters:**
callback - A callback to be called when this Promise is resolved. Must not be null.

**Returns:**
This Promise.

---

### then

```java
Promise<R> then(Success<? super T, ? extends R> success,
                Failure failure)
```

Chain a new Promise to this Promise with Success and Failure callbacks.

The specified Success callback is called when this Promise is successfully resolved and the specified Failure callback is called when this Promise is resolved with a failure.

This method returns a new Promise which is chained to this Promise. The returned Promise must be resolved when this Promise is resolved after the specified Success or Failure callback is executed. The result of the executed callback must be used to resolve the returned Promise. Multiple calls to this method can be used to create a chain of promises which are resolved in sequence.

If this Promise is successfully resolved, the Success callback is executed and the result Promise, if any, or thrown exception is used to resolve the returned Promise from this method. If this Promise is resolved with a failure, the Failure callback is executed and the returned Promise from this method is failed.

This method may be called at any time including before and after this Promise has been resolved.

Resolving this Promise happens-before any registered callback is called. That is, in a registered callback, `isDone()` must return true and `getValue()` and `getFailure()` must not block.
Interface VoidMethodCall

Parameters:
success - A Success callback to be called when this Promise is successfully resolved. May be null if no Success callback is required. In this case, the returned Promise must be resolved with the value null when this Promise is successfully resolved.
failure - A Failure callback to be called when this Promise is resolved with a failure. May be null if no Failure callback is required.

Returns:
A new Promise which is chained to this Promise. The returned Promise must be resolved when this Promise is resolved after the specified Success or Failure callback, if any, is executed.

```
then
Promise<R> then (Success<? super T, ? extends R> success)

Chain a new Promise to this Promise with a Success callback.

This method performs the same function as calling then(Success, Failure) with the specified Success callback and null for the Failure callback.

Type Parameters:
R - The value type associated with the returned Promise.

Parameters:
success - A Success callback to be called when this Promise is successfully resolved. May be null if no Success callback is required. In this case, the returned Promise must be resolved with the value null when this Promise is successfully resolved.

Returns:
A new Promise which is chained to this Promise. The returned Promise must be resolved when this Promise is resolved after the specified Success, if any, is executed.

See Also:
then(Success, Failure)
```

```
filter
Promise<T> filter (Predicate<? super T> predicate)

Filter the value of this Promise.

If this Promise is successfully resolved, the returned Promise will either be resolved with the value of this Promise if the specified Predicate accepts that value or failed with a NoSuchElementException if the specified Predicate does not accept that value. If the specified Predicate throws an exception, the returned Promise will be failed with the exception.

If this Promise is resolved with a failure, the returned Promise will be failed with that failure.

This method may be called at any time including before and after this Promise has been resolved.

Parameters:
predicate - The Predicate to evaluate the value of this Promise. Must not be null.

Returns:
A Promise that filters the value of this Promise.
```

```
map
Promise<R> map (Function<? super T, ? extends R> mapper)

Map the value of this Promise.

If this Promise is successfully resolved, the returned Promise will be resolved with the value of specified Function as applied to the value of this Promise. If the specified Function throws an exception, the returned Promise will be failed with the exception.
```
If this Promise is resolved with a failure, the returned Promise will be failed with that failure.

This method may be called at any time including before and after this Promise has been resolved.

**Type Parameters:**

- The value type associated with the returned Promise.

**Parameters:**

- mapper - The Function that will map the value of this Promise to the value that will be used to resolve the returned Promise. Must not be `null`.

**Returns:**

A Promise that returns the value of this Promise as mapped by the specified Function.

---

### flatMap

```java
Promise<R> flatMap(Function<? super T, Promise<? extends R>> mapper)
```

FlatMap the value of this Promise.

If this Promise is successfully resolved, the returned Promise will be resolved with the Promise from the specified Function as applied to the value of this Promise. If the specified Function throws an exception, the returned Promise will be failed with the exception.

If this Promise is resolved with a failure, the returned Promise will be failed with that failure.

This method may be called at any time including before and after this Promise has been resolved.

**Type Parameters:**

- The value type associated with the returned Promise.

**Parameters:**

- mapper - The Function that will flatMap the value of this Promise to a Promise that will be used to resolve the returned Promise. Must not be `null`.

**Returns:**

A Promise that returns the value of this Promise as mapped by the specified Function.

---

### recover

```java
Promise<T> recover(Function<Promise<?>,? extends T> recovery)
```

Recover from a failure of this Promise with a recovery value.

If this Promise is successfully resolved, the returned Promise will be resolved with the value of this Promise.

If this Promise is resolved with a failure, the specified Function is applied to this Promise to produce a recovery value.

- If the recovery value is not `null`, the returned Promise will be resolved with the recovery value.
- If the recovery value is `null`, the returned Promise will be failed with the failure of this Promise.
- If the specified Function throws an exception, the returned Promise will be failed with that exception.

To recover from a failure of this Promise with a recovery value of `null`, the `recoverWith(Function)` method must be used. The specified Function for `recoverWith(Function)` can return `Promises.resolved(null)` to supply the desired `null` value.

This method may be called at any time including before and after this Promise has been resolved.

**Parameters:**

- recovery - If this Promise resolves with a failure, the specified Function is called to produce a recovery value to be used to resolve the returned Promise. Must not be `null`.

**Returns:**

A Promise that resolves with the value of this Promise or recovers from the failure of this Promise.
recoverWith

\[Promise<T>\]
\texttt{recoverWith}(\texttt{Function<Promise<?>, Promise<? extends T>> recovery)}

Recover from a failure of this Promise with a recovery Promise.

If this Promise is successfully resolved, the returned Promise will be resolved with the value of this Promise.

If this Promise is resolved with a failure, the specified Function is applied to this Promise to produce a recovery Promise.

- If the recovery Promise is not \texttt{null}, the returned Promise will be resolved with the recovery Promise.
- If the recovery Promise is \texttt{null}, the returned Promise will be failed with the failure of this Promise.
- If the specified Function throws an exception, the returned Promise will be failed with that exception.

This method may be called at any time including before and after this Promise has been resolved.

**Parameters:**

- \texttt{recovery} - If this Promise resolves with a failure, the specified Function is called to produce a recovery Promise to be used to resolve the returned Promise. Must not be \texttt{null}.

**Returns:**

A Promise that resolves with the value of this Promise or recovers from the failure of this Promise.

fallbackTo

\[Promise<T>\]
\texttt{fallbackTo}(\texttt{Promise<? extends T> fallback)}

Fall back to the value of the specified Promise if this Promise fails.

If this Promise is successfully resolved, the returned Promise will be resolved with the value of this Promise.

If this Promise is resolved with a failure, the successful result of the specified Promise is used to resolve the returned Promise. If the specified Promise is resolved with a failure, the returned Promise will be failed with the failure of this Promise rather than the failure of the specified Promise.

This method may be called at any time including before and after this Promise has been resolved.

**Parameters:**

- \texttt{fallback} - The Promise whose value will be used to resolve the returned Promise if this Promise resolves with a failure. Must not be \texttt{null}.

**Returns:**

A Promise that returns the value of this Promise or falls back to the value of the specified Promise.
public class Promises
extends Object

Static helper methods for Promises.

ThreadSafe

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Promise&lt;List&lt;T&gt;&gt; all(Collection&lt;? extends T&gt; promises)</td>
<td>Create a new Promise that is a latch on the resolution of the specified Promises.</td>
<td>37</td>
</tr>
<tr>
<td>static Promise&lt;List&lt;T&gt;&gt; all(Promise&lt;? extends T&gt;... promises)</td>
<td>Create a new Promise that is a latch on the resolution of the specified Promises.</td>
<td>37</td>
</tr>
<tr>
<td>static Promise&lt;T&gt; failed(Throwable failure)</td>
<td>Create a new Promise that has been resolved with the specified failure.</td>
<td>36</td>
</tr>
<tr>
<td>static Promise&lt;T&gt; resolved(T value)</td>
<td>Create a new Promise that has been resolved with the specified value.</td>
<td>36</td>
</tr>
</tbody>
</table>

Method Detail

resolved

public static Promise<T> resolved(T value)

Create a new Promise that has been resolved with the specified value.

Type Parameters:

T - The value type associated with the returned Promise.

Parameters:

value - The value of the resolved Promise.

Returns:

A new Promise that has been resolved with the specified value.

failed

public static Promise<T> failed(Throwable failure)

Create a new Promise that has been resolved with the specified failure.

Type Parameters:

T - The value type associated with the returned Promise.

Parameters:

failure - The failure of the resolved Promise. Must not be null.

Returns:

A new Promise that has been resolved with the specified failure.
public static <List<T>> all(Collection<Promise<S>> promises)

Create a new Promise that is a latch on the resolution of the specified Promises.

The new Promise acts as a gate and must be resolved after all of the specified Promises are resolved.

Type Parameters:

T - The value type of the List value associated with the returned Promise.
S - A subtype of the value type of the List value associated with the returned Promise.

Parameters:

promises - The Promises which must be resolved before the returned Promise must be resolved. Must not be null.

Returns:

A Promise that is resolved only when all the specified Promises are resolved. The returned Promise will be successfully resolved, with a List of the values in the order of the specified Promises, if all the specified Promises are successfully resolved. The List in the returned Promise is the property of the caller and is modifiable. The returned Promise will be resolved with a failure of FailedPromisesException if any of the specified Promises are resolved with a failure. The failure FailedPromisesException must contain all of the specified Promises which resolved with a failure.

public static <List<T>> all(Promise<? extends T>... promises)

Create a new Promise that is a latch on the resolution of the specified Promises.

The new Promise acts as a gate and must be resolved after all of the specified Promises are resolved.

Type Parameters:

T - The value type associated with the specified Promises.

Parameters:

promises - The Promises which must be resolved before the returned Promise must be resolved. Must not be null.

Returns:

A Promise that is resolved only when all the specified Promises are resolved. The returned Promise will be successfully resolved, with a List of the values in the order of the specified Promises, if all the specified Promises are successfully resolved. The List in the returned Promise is the property of the caller and is modifiable. The returned Promise will be resolved with a failure of FailedPromisesException if any of the specified Promises are resolved with a failure. The failure FailedPromisesException must contain all of the specified Promises which resolved with a failure.
Interface Success

Type Parameters:

- **T** - The value type of the resolved Promise passed as input to this callback.
- **R** - The value type of the returned Promise from this callback.

@org.osgi.annotation.versioning.ConsumerType
public interface Success

Success callback for a Promise.

A Success callback is registered with a Promise using the Promise.then(Success) method and is called if the Promise is resolved successfully.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

ThreadSafe

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Promise&lt;R&gt; call(Promise&lt;T&gt; resolved)</td>
<td>Success callback for a Promise.</td>
<td>38</td>
</tr>
</tbody>
</table>

Method Detail

**call**

```java
call(Promise<T> resolved)
```

Success callback for a Promise.

This method is called if the Promise with which it is registered resolves successfully.

In the remainder of this description we will refer to the Promise returned by this method as the returned Promise and the Promise returned by Promise.then(Success) when this Success callback was registered as the chained Promise.

If the returned Promise is null then the chained Promise will resolve immediately with a successful value of null. If the returned Promise is not null then the chained Promise will be resolved when the returned Promise is resolved.

**Parameters:**

- resolved - The successfully resolved Promise.

**Returns:**

The Promise to use to resolve the chained Promise, or null if the chained Promise is to be resolved immediately with the value null.

**Throws:**

- Exception - The chained Promise will be failed with the thrown exception.
7.2 The Async Services API
## Package Summary

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.service.async</td>
<td>Asynchronous Services Package Version 1.0.</td>
<td>20</td>
</tr>
</tbody>
</table>
Package org.osgi.service.async

Asynchronous Services Package Version 1.0.

See: Description

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Async</td>
<td>The Asynchronous Execution Service.</td>
</tr>
<tr>
<td>AsyncDelegate</td>
<td>This interface is used by services to allow them to optimize Asynchronous calls where they are capable of executing more efficiently.</td>
</tr>
</tbody>
</table>

Package org.osgi.service.async Description

Asynchronous Services 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.async; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.async; version="[1.0,1.1)"
```
The Asynchronous Execution Service. This can be used to make asynchronous invocations on OSGi services and objects through the use of a mediator object.

Typical usage:

```java
Async async = ctx.getService(asyncRef);
ServiceReference<MyService> ref = ctx.getServiceReference(MyService.class);
MyService asyncMediator = async.mediate(ref);
Promise<BigInteger> result = async.call(asyncMediator.getSumOverAllValues());
```

The `org.osgi.util.promise.Promise` API allows callbacks to be made when asynchronous tasks complete, and can be used to chain Promises. Multiple asynchronous tasks can be started concurrently, and will run in parallel if the Async service has threads available.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>call()</code></td>
<td>This method launches the last method call registered by a mediated object as an asynchronous task.</td>
<td>21</td>
</tr>
<tr>
<td><code>call(R r)</code></td>
<td>This method launches the last method call registered by a mediated object as an asynchronous task.</td>
<td>21</td>
</tr>
<tr>
<td><code>execute()</code></td>
<td>This method should be used by clients in preference to <code>call()</code> and <code>call(Object)</code> when no callbacks, or other features of <code>org.osgi.util.promise.Promise</code>, are needed.</td>
<td>22</td>
</tr>
<tr>
<td><code>T mediate(org.osgi.framework.ServiceReference&lt;T&gt; target)</code></td>
<td>Create a mediator for the given service.</td>
<td>21</td>
</tr>
<tr>
<td><code>T mediate(T target)</code></td>
<td>Create a mediator for the given object.</td>
<td>21</td>
</tr>
</tbody>
</table>

### Method Detail

**mediate**

`T mediate(T target)`

Create a mediator for the given object. The mediator is a generated object that registers the method calls made against it. The registered method calls can then be run asynchronously using either the `call(Object)` or `call()` method.

The values returned by method calls made on a mediated object should be ignored.

Normal usage:

```java
I i = async.mediate(s);
Promise<String> p = async.call(i.foo());
```

**Parameters:**

- `target` - The service object
Interface Async

Returns:
A mediator for the service object

mediate

T mediate(org.osgi.framework.ServiceReference<T> target)

Create a mediator for the given service. The mediator is a generated object that registers the method calls made against it. The registered method calls can then be run asynchronously using either the call(Object) or call() method.

The values returned by method calls made on a mediated object should be ignored.

This method differs from mediate(Object) in that it can track the availability of the backing service. This is recommended as the preferred option for mediating OSGi services as asynchronous tasks may not start executing until some time after they are requested. Tracking the validity of the org.osgi.framework.ServiceReference for the service ensures that these tasks do not proceed with an invalid object.

Normal usage:

I i = async.mediate(s);
Promise<String> p = async.call(i.foo());

Parameters:
  target - The service object

Returns:
  A mediator for the service object

call

org.osgi.util.promise.Promise<R> call(R r)

This method launches the last method call registered by a mediated object as an asynchronous task. The result of the task can be obtained using the returned promise.

Typically the parameter for this method will be supplied inline like this:

I i = async.mediate(s);
Promise<String> p = async.call(i.foo());

Parameters:
  r - the return value of the mediated call, used for type information

Returns:
  a Promise which can be used to retrieve the result of the asynchronous execution

call

org.osgi.util.promise.Promise<?> call()

This method launches the last method call registered by a mediated object as an asynchronous task. The result of the task can be obtained using the returned promise.

Generally it is preferrable to use call(Object), like this:

I i = async.mediate(s);
Promise<String> p = async.call(i.foo());

However this pattern does not work for void methods. Void methods can therefore be handled like this:
I i = async.mediate(s);
i.voidMethod()
Promise<?> p = async.call();

**Returns:**
a Promise which can be used to retrieve the result of the asynchronous execution

---

## execute

**void execute()**

This method should be used by clients in preference to `call()` and `call(Object)` when no callbacks, or other features of `org.osgi.util.promise.Promise`, are needed.

This method launches the last method call registered by a mediated object as an asynchronous task. The task runs as a "fire and forget" process, and there will be no notification of its success or failure.

Typically this method is used like `call()`:

```java
I i = async.mediate(s);
i.someMethod()
Promise<?> p = async.call();
```

The advantage of the `execute()` method is that it allows for greater optimisation of the underlying asynchronous execution. Clients are therefore likely to see better performance when using this method compared to using `call()` and discarding the return value.
Interface AsyncDelegate

This interface is used by services to allow them to optimize Asynchronous calls where they are capable of executing more efficiently. This may mean that the service has access to its own thread pool, or that it can delegate work to a remote node, or act in some other way to reduce the load on the Asynchronous Services implementation when making an asynchronous call.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.util.promise.Promise&lt;?&gt; async (Method m, Object[] args)</td>
<td>This method can be used by clients, or the Async service, to optimize Asynchronous execution of methods.</td>
</tr>
<tr>
<td>boolean execute (Method m, Object[] args)</td>
<td>This method can be used by clients, or the Async service, to optimize Asynchronous execution of methods.</td>
</tr>
</tbody>
</table>

Method Detail

async

org.osgi.util.promise.Promise<?> async (Method m, Object[] args)
  throws Exception

This method can be used by clients, or the Async service, to optimize Asynchronous execution of methods. When called, the AsyncDelegate should execute the supplied method using the supplied arguments asynchronously, returning a promise that can be used to access the result. If the method cannot be executed asynchronously by the delegate then it should return null.

Parameters:
- m - the method that should be asynchronously executed
- args - the arguments that should be used to invoke the method

Returns:
A promise representing the asynchronous result, or null if this method cannot be asynchronously invoked by the AsyncDelegate.

Throws:
Exception - An exception should be thrown only if there was an serious error that prevented the asynchronous call from starting, for example the supplied method does not exist on this object. Exceptions should not be thrown to indicate that the call does not support asynchronous invocation, instead the AsyncDelegate should return null. Exceptions should also not be thrown to indicate a failure from the execution of the underlying method, this should be handled by failing the returned promise.

execute

boolean execute (Method m, Object[] args)
  throws Exception

This method can be used by clients, or the Async service, to optimize Asynchronous execution of methods. When called, the AsyncDelegate should execute the supplied method using the supplied arguments asynchronously. This method differs from async(Method, Object[]) in that it does not return a promise. This method therefore allows the implementation to perform more aggressive optimisations because the end result of the invocation does not need to be returned to the client. If the method cannot be executed asynchronously by the delegate then it should return false.
Interface AsyncBuilder

Parameters:
- m - the method that should be asynchronously executed
- args - the arguments that should be used to invoke the method

Returns:
true if the asynchronous execution request has been accepted, or false if this method cannot be asynchronously invoked by the AsyncDelegate.

Throws:
Exception - An exception should be thrown only if there was an serious error that prevented the asynchronous call from starting, for example the supplied method does not exist on this object. Exceptions should not be thrown to indicate that the call does not support asynchronous invocation, instead the AsyncDelegate should return false. Exceptions should also not be thrown to indicate a failure from the execution of the underlying method.

8 Considered Alternatives

Initially this RFC used a builder model to create aggregate tasks. With the addition of the Promise API this is no longer necessary.

8.1.1 Completing tasks

Once a client has established a context then they can “complete” building the task using either the launch() method or asPromise() method. The launch() method is used for “Fire and Forget” style invocations, while the asPromise() method returns a java.util.concurrent.Future, typed appropriately for the return type of the asynchronous invocation.

Once one of the “completion” methods has been invoked the asynchronous task should begin executing. The real service object should be obtained by the Async Service implementation by using the client's BundleContext to call getService() on the ServiceReference used to create the async mediator.

Example – Fire and Forget:

```java
private Async asyncService;
private ServiceReference<List> listRef;
@Reference
synchronized void setAsync(Async async) {
    asyncService = async;
}
@Reference(service = List.class)
synchronized void setList(ServiceReference<List> list) {
    listRef = list;
}
public synchronized void doStuff() {
    List mediator = asyncService.createAsyncMediator(ref);
    asyncService.build(mediator.contains("badEntry")).launch();
}
```

Example – With a promise:
private Async asyncService;
private ServiceReference<List> listRef;

@synchronized void setAsync(Async async) {
    asyncService = async;
}

@synchronized void setList(ServiceReference<List> list) {
    listRef = list;
}

public synchronized void doStuff() {
    List mediator = asyncService.createAsyncMediator(ref);
    Future<Boolean> futureResult = asyncService
        .build(mediator.contains("badEntry")).asPromise();
    ...
}

The Future returned by the asPromise() method must obey the Java contracts for Futures. It must:

- be thread safe,
- implement a cancel method that should make a best-effort to cancel or prevent the execution of the asynchronous invocation.
- provide a blocking get methods which returns the result of the execution
- throw an ExecutionException from the get methods, wrapping the real failure, if the asynchronous invocation threw an Exception
- throw a CancellationException from the get methods if the Future was cancelled
- provide methods for determining whether a Future is finished, and whether it has been cancelled.

### 8.1.2 Registering Callbacks

Having established a context, a client may register callbacks with the asynchronous services implementation.

There are three kinds of callback:

- Success Callbacks – these are called with the result of the asynchronous invocation, if it returned normally. Implements the org.osgi.service.async.SuccessCallback interface.
- Failure Callbacks – these are called with the exception thrown by the asynchronous invocation, if it returned exceptionally. Implements the org.osgi.service.async.FailureCallback interface.
- Completion Callbacks – these are always called after the asynchronous invocation has completed. Completion Callbacks will be made after any success or failure callbacks have been made. Implements the org.osgi.service.async.CompletionCallback interface.

All three callback interfaces follow the Single Abstract Method principle, which means that they are able to be substituted for lambda expressions in Java 8.

Callbacks can be registered against the asynchronous invocation represented by the current context using the onSuccess(), onFailure() and onCompletion() methods:

Example:

```java
private Async asyncService;
private ServiceReference<List> listRef;
@synchronized void setAsync(Async async) {
```
asyncService = async;
}

@Reference(service = List.class)
synchronized void setList(ServiceReference<List> list) {
    listRef = list;
}

public synchronized void doStuff() {
    List mediator = asyncService.createAsyncMediator(ref);
    asyncService.build(mediator.remove("badEntry"))
        .onSuccess(new MySuccessCallback())
        .onFailure(new MyFailureCallback())
        .onCompletion(new MyCompletionCallback()).launch();
}

Callbacks may also be used in conjunction with the asPromise() method.

8.2 More complex tasks
8.2.1 Running in Parallel

Most commonly asynchronous invocations are used to run multiple tasks in parallel. This can be achieved using the parallel() method to establish a new context. Once a new context has been established any calls to onSuccess(), onFailure() or onCompletion() will be applied to the new context.

Any parallel asynchronous invocations in an Asynchronous task are eligible to be run in parallel with the preceding asynchronous invocation. Note that this does not mean that the tasks will definitely run in parallel, for example there may be insufficient worker threads available to run additional tasks.

Example:

private Async asyncService;
private ServiceReference<List> listRef;
@Reference
synchronized void setAsync(Async async) {
    asyncService = async;
}

@Reference(service = List.class)
synchronized void setList(ServiceReference<List> list) {
    listRef = list;
}

public synchronized void doStuff() {
    List mediator = asyncService.createAsyncMediator(ref);
    asyncService.build(mediator.contains("badEntry"))
        .onSuccess(new MySuccessCallback()) //Applies to "badEntry"
        .parallel(mediator.contains("goodEntry"))
        .onSuccess(new AnotherSuccessCallback()) // Applies to "goodEntry"
        .launch();
}

8.2.2 Running sequentially

Sometimes asynchronous invocations have an implicit ordering requirement. In this case a single task can be created that only starts running an invocation after the previous invocation has successfully completed. This is accomplished by using the then() method to establish a new context. This works in the same way as the parallel() method, but the new asynchronous invocation is only eligible to run after the previous task returns.

Example:

private Async asyncService;
private ServiceReference<List> listRef;
@Reference
synchronized void setAsync(Async async) {
    asyncService = async;
}
8.2.3 Waiting for previous invocations

Sometimes it is not enough to wait for a single asynchronous invocation to complete, and instead you need to wait for a group of tasks to complete before continuing. In this case the `afterAll()` method establishes a new asynchronous invocation context that is only eligible to execute after all previous asynchronous invocations have completed.

Example:

```java
private Async asyncService;
private ServiceReference<List> listRef;

@synchronized void setAsync(Async async) {
    asyncService = async;
}

@synchronized void setList(ServiceReference<List> list) {
    listRef = list;
}

public synchronized void doStuff() {
    List mediator = asyncService.createAsyncMediator(ref);
    asyncService.build(mediator.contains("badEntry")
        .then(mediator.contains("badEntry"))
        .onSuccess(new AnotherSuccessCallback()) // Applies to contains()
        .launch();
}
```

8.2.4 Receiving multiple promises

Some clients that issue multi-invocation asynchronous tasks will wish to consume the results of their asynchronous invocations using the `Future` API. Therefore the `org.osgi.service.async.AsyncCompleter` interface defines an `asPromises()` method that completes the asynchronous task, returning a `List` of `Future` objects, representing the completions of each asynchronous invocation in the task. The order of the objects in the `List` is the same as the order in which the client created the asynchronous invocation contexts that make up the task.

Example:

```java
private Async asyncService;
private ServiceReference<List> listRef;

@synchronized void setAsync(Async async) {
    asyncService = async;
}

@synchronized void setList(ServiceReference<List> list) {
    listRef = list;
}
```
8.2.5 Registering aggregate callbacks

It can be useful for clients to receive notifications about the overall progress of an asynchronous task, rather than about its individual elements. To support this use case the `AsynchronousBuilder` interface declares the `andFinally()` method, which establishes a special context. This context represents the entire asynchronous task, not just a single asynchronous invocation within the task. This means that any callbacks registered using `onSuccess()` or `onCompletion()` will be called when the entire task has completed. Success callbacks will be called with a null argument. Failure callbacks registered with `onFailure()` will receive callbacks immediately when any of the asynchronous invocations that make up the asynchronous task fail. This means that the failure callback may be called multiple times, possibly concurrently.

It is supported to use aggregate callbacks as well as callbacks on individual asynchronous invocations.

Example:

```java
private Async asyncService;
private ServiceReference<List> listRef;
@Reference
synchronized void setAsync(Async async) {
    asyncService = async;
}

@Reference(service = List.class)
synchronized void setList(ServiceReference<List> list) {
    listRef = list;
}

public synchronized void doStuff() {
    List mediator = asyncService.createAsyncMediator(ref);
    List<Future<?>> promises = asyncService.build(mediator.contains("badEntry"))
        .parallel(mediator.contains("goodEntry"))
        .parallel(mediator.contains("anotherEntry"))
        .andFinally().onCompletion(new AllChecksFinished())
        .asPromises(); //List is in the order 'bad, good, another'
    ...
}
```

9 Security Considerations

Asynchronous Services implementations must be careful to avoid elevating the privileges of client bundles when calling services asynchronously. This means that the implementation must:

- Use the client bundle to load interfaces when generating the asynchronous mediator. This prevents clients from gaining access to interfaces they would not normally be permitted to import.
Use the client's bundle context when retrieving the target service. This prevents the client bundle from being able to make calls on a service object that they would normally be forbidden from obtaining.

Further security considerations can be addressed using normal OSGi security rules, access to the Async service can be controlled using ServicePermission[Async, GET].

## 10 Document Support

### 10.1 References


### 10.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Tim Ward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Paremus Ltd</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:tim.ward@paremus.com">tim.ward@paremus.com</a></td>
</tr>
</tbody>
</table>

### 10.3 Acronyms and Abbreviations

### 10.4 End of Document
Abstract

This RFC introduces annotations for the Metatype specification which can be use to annotate Java types so that tools can generate Meta Type Resources from the type declaration.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

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

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such as 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
I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

<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 License</td>
<td>2</td>
</tr>
<tr>
<td>0.2 Trademarks</td>
<td>3</td>
</tr>
<tr>
<td>0.3 Feedback</td>
<td>3</td>
</tr>
<tr>
<td>0.4 Table of Contents</td>
<td>3</td>
</tr>
<tr>
<td>0.5 Terminology and Document Conventions</td>
<td>4</td>
</tr>
<tr>
<td>0.6 Revision History</td>
<td>4</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>5</td>
</tr>
<tr>
<td>2 Application Domain</td>
<td>5</td>
</tr>
<tr>
<td>3 Problem Description</td>
<td>6</td>
</tr>
<tr>
<td>4 Requirements</td>
<td>6</td>
</tr>
<tr>
<td>5 Technical Solution</td>
<td>6</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>6</td>
</tr>
<tr>
<td>5.2 @ObjectClassDefinition</td>
<td>8</td>
</tr>
<tr>
<td>5.3 @AttributeDefinition</td>
<td>9</td>
</tr>
<tr>
<td>5.4 @Designate</td>
<td>10</td>
</tr>
<tr>
<td>5.5 @Option</td>
<td>10</td>
</tr>
<tr>
<td>5.6 @Icon</td>
<td>10</td>
</tr>
<tr>
<td>5.7 Other Changes</td>
<td>10</td>
</tr>
<tr>
<td>5.7.1 Bug 2436</td>
<td>10</td>
</tr>
<tr>
<td>5.7.2 Bug 2540</td>
<td>10</td>
</tr>
<tr>
<td>5.7.3 List replaces Vector</td>
<td>11</td>
</tr>
</tbody>
</table>

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.
0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

0.6 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>2013-11-19</td>
<td>Initial draft. BJ Hargrave, IBM</td>
</tr>
<tr>
<td>2nd</td>
<td>2013-11-20</td>
<td>Updated after feedback from Peter Kriens. Replaced Designate annotation with designate and designateFactory elements on the ObjectClassDefinition annotation. Also added icon element (and Icon annotation) to ObjectClassDefinition. BJ Hargrave, IBM</td>
</tr>
<tr>
<td>3rd</td>
<td>2013-11-21</td>
<td>Updates from CPEG meeting. Remove id element from AttributeDefinition. Change annotations to CLASS retention. Rename designate elements to pid. Allow negative cardinality values to mean List. Allow ObjectClassDefinition to be applied to interfaces. Update Meta Type spec to replace use of Vector with List. BJ Hargrave, IBM</td>
</tr>
<tr>
<td>4th</td>
<td>2013-12-05</td>
<td>Accepted changes after review at CPEG meeting. BJ Hargrave, IBM</td>
</tr>
<tr>
<td>5th</td>
<td>2014-06-25</td>
<td>Added Designate annotation to replace pid/factoryPid elements of ObjectClassDefinition. BJ Hargrave, IBM</td>
</tr>
</tbody>
</table>
1 Introduction

The Metatype specification defines a Meta Type Resource format which can be used by Meta Type Service implementations. These resources are XML documents which conform to the Meta Type Resource XML Schema. RFC 190 introduces annotation configuration types to DS so that developers can access their configuration (component properties) in a type safe way. Since the configuration is now describable as a Java type, this RFC will also allow the type to document the Meta Type information about the configuration so that tools can generate Meta Type Resources from the Java type.

2 Application Domain

OSGi has long had the Meta Type specification which defines meta type information for configurations which are stored in Configuration Admin service. The Meta Type definitions are useful by GUIs to allow users to define actual configurations by providing information about the expected data types and values including localized information for a GUI. Meta Type specification also defines a Meta Type Resource format which is an XML document that can be contained in a bundle and processed by the Meta Type service.

Declarative Services uses configurations from Configuration Admin service as component properties for components. RFC 190 is updating DS to allow the component properties to be “shaped” into annotation types to provide components type-safe access to their component properties.

RFC 179 “DS Updates for Configurable” is an RFC which is no longer being worked but which contains the seed of the design now being using in RFC 190 for the configuration annotation types. RFC 179 is based upon RFC 178 “Configurable” which includes design ideas on annotations of these types for Meta Type support.

Bnd has also provided support for Meta Type annotations. See http://www.aqute.biz/Bnd/MetaType. The Meta.OCD and Meta.AD annotations were inputs to RFC 178.
3 Problem Description

Writing Meta Type Resource documents requires the programmer to author an XML document which both conforms to the Meta Type XML schema and accurately reflects the data and data types in the configuration. The programmer must keep changes to the program using the configuration and the XML document in sync. This can be difficult during refactoring and hard to validate during testing to avoid allowing errors from being propagated.

4 Requirements

MTA-0100 – Meta Type resource information must be able to be described in Java source code. This allows for compiler checking of types and refactoring support.

MTA-0200 – Must be able to mark a configuration annotation type (from RFC 190) as a source for Meta Type information.

MTA-0300 – Defaults for meta type information must be derivable from the marked type.

MTA-0400 – The programmer must be able to supply meta type information to override the defaults.

MTA-0500 – Tools must be able to process the meta type information specified in the source so that Meta Type Resource XML documents can be automatically generated.

MTA-0600 – Meta type information from the source files must also be present in the generated class files so tools do not need to process the source files.

5 Technical Solution

5.1 Introduction

Annotations are defined that can applied to the configuration annotation types from RFC 190. An example from RFC 190:

@interface Config {
boolean enabled() default true;
String[] names() default "a", "b";
String topic() default MyComponent.DEFAULT_TOPIC_PREFIX + "/topic";
}

@Component
public class MyComponent {
    static final String DEFAULT_TOPIC_PREFIX = "topic.prefix";
    protected void activate(Config configuration) {
        String t = configuration.topic();
    }
}

In this example, the Config annotation type is used as a configuration type which is used by the activate method. The Config type describes the "shape" of the configuration and can be used to also describe the meta type information. If we annotate the Config type with the new ObjectClassDefinition annotation,

@ObjectClassDefinition
@interface Config {
    boolean enabled() default true;
    String[] names() default 
cción ["a", "b"];
    String topic() default MyComponent.DEFAULT_TOPIC_PREFIX + "/topic";
}

@Component
@Designate(ocd = Config.class)
public class MyComponent {
    static final String DEFAULT_TOPIC_PREFIX = "topic.prefix";
    protected void activate(Config configuration) {
        String t = configuration.topic();
    }
}

a tool (like bnd) processing the bundle can automatically generate a Meta Type Resource XML document from the information in the Config type. The main purpose of these annotations is to generate Meta Type Resource XML documents from the configuration annotations.

In this larger example:

@ObjectClassDefinition(localization = "OSGI-INF/l10n/test",
description = "%test.description",
name = "%test.name"
icon = @Icon(resource = "icon/test-32.png", size = 32))
public @interface Test {
    @AttributeDefinition(type = AttributeType.PASSWORD,
description = "%test.password.description",
name = "%test.password.name")
public String _password();

    @AttributeDefinition(options = {
        @Option(label = "%strategic", value = "strategic"),
        @Option(label = "%principal", value = "principal"),
        @Option(label = "%contributing", value = "contributing")
    },
    defaultValue = "contributing",
description = "%test.membertype.description",

name = "%test.membertype.name")

    public String memberType();

    public String my_prop();
}

@Component(configurationPid = "test.pid")
@Designate(ocd = Config.class)
public class MyComponent {

    static final String DEFAULT_TOPIC_PREFIX = "topic.prefix";

    protected void activate(Config configuration) {
        String t = configuration.topic();
    }
}

we can see more extensive use of the new annotations. ObjectClassDefinition marks the Test type as a configuration type for which a meta type resource should be generated. It further defines meta type information including the description and name which are to be localized using the specified resource as well as an icon resource. AttributeDefinition marks elements of the Test type to provide meta type information. If meta type information is not provided by the annotation declaration, default information must be generated from the annotated type. The Designate annotation on the components connects the ObjectClassDefinition defined above with the pid of the component.

This RFC is tied to RFC 190 in that the annotations defined here are to be applied to the configuration annotation types defined by RFC 190.

5.2 @ObjectClassDefinition

The ObjectClassDefinition annotation is applied to a type to mark it for processing into a Meta Type Resource XML document.

The ObjectClassDefinition annotation can be applied without defining any element values as defaults can be generated from the annotated type. The following elements are defined:

- **name** – (String) A human readable name of the object, can be localized if it starts with a % sign. The default is a string derived from the id where _, $, or camel casing is used to provide spaces. The name becomes the value of the name attribute of the OCD element in the generated Meta Type Resource XML document.

- **id** – (String) The id of the object, the default is the fully qualified name of the type with a $ as separator for nested classes. This is not to be confused with a PID which can be specified by the pid or factoryPid element. The id becomes the value of the id attribute of the OCD element in the generated Meta Type Resource XML document.

- **localization** – (String) The localization resource of the object. This refers to a resource property entry in the bundle that can be augmented with locale information. The default is the fully qualified name of the class prefixed by “OSGI-INF/l10n/”. The localization becomes the value of the localization attribute of the OCD element in the generated Meta Type Resource XML document.

- **description** – (String) A human readable description that can be localized when it starts with %. Default is the empty string. The description becomes the value of the description attribute of the OCD element in the generated Meta Type Resource XML document.
• pid – (String[]) The PIDs associated with the ObjectClassDefinition. The default is no associated PIDs. The pid information becomes a set of Designate elements for each pid which reference the OCD element in the generated Meta Type Resource XML document.

• factoryPid – (String[]) The factory PIDs associated with the ObjectClassDefinition. The default is no associated factory PIDs. The factory pid information becomes a set of Designate elements for each factoryPid which reference the OCD element in the generated Meta Type Resource XML document.

• icon – (Icon[]) Specify icons (resource name and size). The default is no icon information. The icon information becomes a set of Icon elements of the OCD element in the generated Meta Type Resource XML document.

Each method of the type annotated by ObjectClassDefinition is mapped to an AD child element of the OCD element in the generated Meta Type Resource XML document. The AttributeDefinition annotation only needs to be applied if values other than the defaults are desired.

5.3 @AttributeDefinition

The AttributeDefinition annotation is an optional annotation which can applied to elements in a configuration annotation type annotated by ObjectClassDefinition. Each element of the configuration annotation type annotated by ObjectClassDefinition is mapped to an AD child element of the OCD element in the generated Meta Type Resource XML document. The AttributeDefinition annotation only needs to be applied if values other than the defaults are desired. The id of the AttributeDefinition is generated from the annotated element name as specified in RFC 190 section 5.6.2 (e.g. removal of dollar sign and converting underscore to dot). The id becomes the value of the id attribute of the AD element in the generated Meta Type Resource XML document and is used as the name of the configuration property. The following elements are defined:

• name – (String) A human readable name of the attribute, can be localized if it starts with a % sign. The default is a string derived from the method name where _, $, or camel casing is used to provide spaces. The name becomes the value of the name attribute of the AD element in the generated Meta Type Resource XML document.

• description – (String) A human readable description that can be localized if it starts with %. Default is the empty string. The description becomes the value of the description attribute of the AD element in the generated Meta Type Resource XML document.

• type – (AttributeType) The type of the attribute. This must be one of the types defined in the Metatype specification. The default is derived from the type of the element. Class and Enum types are mapped to String. Annotation types are not supported. A tool processing the annotation should declare an error during processing in this case. The type is used to select the value of the type attribute of the AD element in the generated Meta Type Resource XML document.

• cardinality - (int) The cardinality of the attribute. The default is 0 if the element is not an array and a large positive number if the element is an array type. If the element is an array type, a negative value can be specified to indicate the property value should be a list instead of an array. The cardinality becomes the value of the cardinality attribute of the AD element in the generated Meta Type Resource XML document.

• min – (String) The minimum value allowed for this attribute. There is no default. The min becomes the value of the min attribute of the AD element in the generated Meta Type Resource XML document.

• max – (String) The maximum value allowed for this attribute. There is no default. The max becomes the value of the max attribute of the AD element in the generated Meta Type Resource XML document.
• defaultValue – (String[]) The default values. The defaultValues are concatenated into a comma delimited list to becomes the value of the default attribute of the AD element in the generated Meta Type Resource XML document.

• required – (boolean) Indicates if this attribute is required. The default is true. The required becomes the value of the required attribute of the AD element in the generated Meta Type Resource XML document.

• options - (Option[]) Specify options (value and optional label). There is only a default if the element type is an Enum or Enum[] in which case the label is the enum element toString() output and the value is the enum element name() output. The options information becomes a set of Option elements of the AD element in the generated Meta Type Resource XML document.

5.4 @Designate
The Designate annotation can be applied to a Declarative Services component class to make the connection between the pid of the component and an ObjectClassDefinition. This annotation must be used on a type that is also annotated with the Declarative Services @Component annotation. The component must only have a single PID which is used for the generated Designate element.

• ocd – (Class) A class which is annotated with the ObjectClassDefinition annotation. The id of the referenced ObjectClassDefinition is used for the ocdref attribute in the generated Designate element.

• factory – (boolean) If false, then the PID value from the annotated component will be used in the attribute of the generated Designate element. If true, then the PID value from the annotated component will be used in the factoryPid attribute of the generated Designate element.

5.5 @Option
The Option annotation is only used for the options element of the AttributeDefinition annotation to allow specifying label/value pair for an AttributeDefinition.

5.6 @Icon
The Icon annotation is only used for the icon element of the ObjectClassDefinition annotation to allow specifying a icon resource/size pair.

5.7 Other Changes
Since this RFC will modify the Meta Type Specification and bump its version to 1.3, we can also pick up some minor Meta Type bugs awaiting a specification version change. The metatype package should also be updated to use the new package and type annotations from RFC 197.

5.7.1 Bug 2436
The schema is fixed to use “Character” instead of “Char” to match the proper Java type name and other OSGi specifications like DS and RSA.

5.7.2 Bug 2540
The schema is modified to allow more flexible ordering of elements.
5.7.3 List replaces Vector

The Meta Type specification will be updated to refer to List instead of Vector. A Vector is a List and List is the more modern type.

6 Javadoc

More Javadoc detail will be added after the first review round of this RFC.
### Package Summary

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.service.metatype.annotations</td>
<td>Metatype Annotations Package Version 1.3.</td>
<td>13</td>
</tr>
</tbody>
</table>
Metatype Annotations Package Version 1.3.

See: Description

### Enum Summary

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AttributeType</strong></td>
<td>Attribute types for the <code>AttributeDefinition</code> annotation.</td>
</tr>
</tbody>
</table>

### Annotation Types Summary

<table>
<thead>
<tr>
<th>Annotation Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AttributeDefinition</strong></td>
<td>AttributeDefinition information for the annotated method.</td>
</tr>
<tr>
<td><strong>Designate</strong></td>
<td>Generate a Meta Type Resource using the annotated Declarative Services component as the PID value for a Designate element.</td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td>Icon information for an <code>ObjectClassDefinition</code>.</td>
</tr>
<tr>
<td><strong>ObjectClassDefinition</strong></td>
<td>Generate a Meta Type Resource using the annotated type as an ObjectClassDefinition.</td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td>Option information for an <code>AttributeDefinition</code>.</td>
</tr>
</tbody>
</table>

Package org.osgi.service.metatype.annotations Description

Metatype Annotations Package Version 1.3.

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.metatype.annotations; version="[1.3,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.metatype.annotations; version="[1.3,1.4)"
```
Annotation Type AttributeDefinition

org.osgi.service.metatype.annotations

@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.METHOD)
public @interface AttributeDefinition

AttributeDefinition information for the annotated method.

Each method of a type annotated by ObjectClassDefinition has an implied AttributeDefinition annotation. This annotation is only used to specify non-default AttributeDefinition information.

The id of this AttributeDefinition is generated from the name of the annotated method. The annotated method name is processed from left to right changing each character as follows:

- A dollar sign (\$) is removed unless it is followed by another dollar sign in which case the two consecutive dollar signs (\$\$) are changed to a single dollar sign.
- A low line (\_) is changed to a full stop (\.) unless is it followed by another low line in which case the two consecutive low lines (\__\_) are changed to a single low line.
- All other characters are unchanged.

This id is the value of the id attribute of the generate AD element and is used as the name of the corresponding configuration property.

This annotation is not processed at runtime. It must be processed by tools and used to generate a Meta Type Resource document for the bundle.

See Also:
- "The AD element of a Meta Type Resource."

### Required Element Summary

<table>
<thead>
<tr>
<th>Element</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>int cardinality</td>
<td>15</td>
</tr>
<tr>
<td>String[] defaultValue</td>
<td>16</td>
</tr>
<tr>
<td>String description</td>
<td>15</td>
</tr>
<tr>
<td>String max</td>
<td>16</td>
</tr>
<tr>
<td>String min</td>
<td>16</td>
</tr>
<tr>
<td>String name</td>
<td>14</td>
</tr>
<tr>
<td>Option[] options</td>
<td>16</td>
</tr>
<tr>
<td>boolean required</td>
<td>16</td>
</tr>
<tr>
<td>AttributeType type</td>
<td>15</td>
</tr>
</tbody>
</table>

### Element Detail

**name**

```java
public abstract String name
```
The human readable name of this AttributeDefinition.

If not specified, the name of this AttributeDefinition is derived from the name of the annotated method. For example, low line (‘_’ \u005F) and dollar sign (‘$’ \u0024) are replaced with space ( ‘ ’ \u0020) and space is inserted between camel case words.

If the name begins with the percent sign (‘%’ \u0025), the name can be localized.

Default: 
See Also: 
"The name attribute of the AD element of a Meta Type Resource."

### description

public abstract String description

The human readable description of this AttributeDefinition.

If not specified, the description of this AttributeDefinition is the empty string.

If the description begins with the percent sign (‘%’ \u0025), the description can be localized.

Default: 
See Also: 
"The description attribute of the AD element of a Meta Type Resource."

### type

public abstract AttributeType type

The type of this AttributeDefinition.

This must be one of the defined attributes types.

If not specified, the type is derived from the return type of the annotated method. Return types of Class and Enum are mapped to STRING. A tool processing the annotation should declare an error for unsupported return types.

Default: AttributeType.STRING
See Also: 
"The type attribute of the AD element of a Meta Type Resource."

### cardinality

public abstract int cardinality

The cardinality of this AttributeDefinition.

If not specified, the cardinality is derived from the return type of the annotated method. For non-array and non-Collection return types, that is a scalar type, the cardinality is 0. For array return types, the cardinality is a large positive value. For Collection return types, the cardinality is a large negative value.

Default: 0
See Also: 
"The cardinality attribute of the AD element of a Meta Type Resource."
min

public abstract String min

The minimum value for this AttributeDefinition.

If not specified, there is no minimum value.

Default: 

See Also:
"The min attribute of the AD element of a Meta Type Resource."

max

public abstract String max

The maximum value for this AttributeDefinition.

If not specified, there is no maximum value.

Default: 

See Also:
"The max attribute of the AD element of a Meta Type Resource."

defaultValue

public abstract String[] defaultValue

The default value for this AttributeDefinition.

If not specified, if the annotated element is part of an annotation and has a default value, then the value is the default value of the annotated element. Otherwise, there is no default value.

Default: 

See Also:
"The default attribute of the AD element of a Meta Type Resource."

required

public abstract boolean required

The required value for this AttributeDefinition.

If not specified, the value is true.

Default: true

See Also:
"The required attribute of the AD element of a Meta Type Resource."

options

public abstract Option[] options
The option information for this AttributeDefinition.

For each specified Option, an Option element is generated for this AttributeDefinition. If not specified, no Option elements will be generated.

**Default:**

```{}
```

**See Also:**

"The Option element of a Meta Type Resource."
public enum AttributeType extends Enum<AttributeType>

Attribute types for the AttributeDefinition annotation.

See Also:
AttributeDefinition.type()
Enum Constant Detail

**STRING**

```java
public static final AttributeType STRING
```

The String type.

Attributes of this type should be stored as String, List<String> or String[] objects, depending on the cardinality value.

**LONG**

```java
public static final AttributeType LONG
```

The Long type.

Attributes of this type should be stored as Long, List<Long> or long[] objects, depending on the AttributeDefinition#cardinality() cardinality value.

**INTEGER**

```java
public static final AttributeType INTEGER
```

The Integer type.

Attributes of this type should be stored as Integer, List<Integer> or int[] objects, depending on the AttributeDefinition#cardinality() cardinality value.

**SHORT**

```java
public static final AttributeType SHORT
```

The Short type.

Attributes of this type should be stored as Short, List<Short> or short[] objects, depending on the AttributeDefinition#cardinality() cardinality value.

**CHARACTER**

```java
public static final AttributeType CHARACTER
```

The Character type.

Attributes of this type should be stored as Character, List<Character> or char[] objects, depending on the AttributeDefinition#cardinality() cardinality value.

**BYTE**

```java
public static final AttributeType BYTE
```

The Byte type.
Attributes of this type should be stored as Byte, List<Byte> or byte[] objects, depending on the AttributeDefinition#cardinality() cardinality value.

DOUBLE

public static final AttributeType DOUBLE

The Double type.

Attributes of this type should be stored as Double, List<Double> or double[] objects, depending on the AttributeDefinition#cardinality() cardinality value.

FLOAT

public static final AttributeType FLOAT

The Float type.

Attributes of this type should be stored as Float, List<Float> or float[] objects, depending on the AttributeDefinition#cardinality() cardinality value.

BOOLEAN

public static final AttributeType BOOLEAN

The Boolean type.

Attributes of this type should be stored as Boolean, List<Boolean> or boolean[] objects depending on AttributeDefinition#cardinality() cardinality.

PASSWORD

public static final AttributeType PASSWORD

The Password type.

Attributes of this type must be stored as String, List<String> or String[] objects depending on cardinality.

A Password must be treated as a String but the type can be used to disguise the information when displayed to a user to prevent it from being seen.

Method Detail

values

public static AttributeType[] values()

valueOf

public static AttributeType valueOf(String name)
**toString**

```java
public String toString()
```

**Overrides:**

```java
toString in class Enum
```
Annotation Type Designate

```java
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.TYPE)
public @interface Designate {

Generate a Meta Type Resource using the annotated Declarative Services component as the PID value for a Designate element.

This annotation must be used on a type that is also annotated with the Declarative Services Component annotation. The component must only have a single PID which is used for the generated Designate element.

This annotation is not processed at runtime. It must be processed by tools and used to generate a Meta Type Resource document for the bundle.

See Also:
"The Designate element of a Meta Type Resource."
```

### Required Element Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>factory</td>
<td>boolean</td>
<td>Specifies whether this Designate is for a factory PID.</td>
<td>22</td>
</tr>
<tr>
<td>ocd</td>
<td>Class&lt;?&gt;</td>
<td>The type of the ObjectClassDefinition for this Designate.</td>
<td>22</td>
</tr>
</tbody>
</table>

### Element Detail

**ocd**

```java
public abstract Class<?> ocd
```

The type of the ObjectClassDefinition for this Designate.

The specified type must be annotated with ObjectClassDefinition.

See Also:
"The ocdref attribute of the Designate element of a Meta Type Resource."

**factory**

```java
public abstract boolean factory
```

Specifies whether this Designate is for a factory PID.

If false, then the PID value from the annotated component will be used in the pid attribute of the generated Designate element. If true, then the PID value from the annotated component will be used in the factoryPid attribute of the generated Designate element.

Default: false

See Also:
"The pid and factoryPid attributes of the Designate element of a Meta Type Resource."
Annotation Type Icon

@Retention(value=RetentionPolicy.CLASS)
@Target(value=\{\})
public @interface Icon

Icon information for an ObjectClassDefinition.

See Also:
ObjectClassDefinition.icon()

<table>
<thead>
<tr>
<th>Required Element Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>resource</strong></td>
<td>String</td>
</tr>
<tr>
<td><strong>size</strong></td>
<td>int</td>
</tr>
</tbody>
</table>

**Element Detail**

**resource**

public abstract String resource

The resource name for this Icon.

The resource is a URL. The resource URL can be relative to the root of the bundle containing the Meta Type Resource.

If the resource begins with the percent sign ('%\u0025'), the resource can be localized.

See Also:
"The resource attribute of the Icon element of a Meta Type Resource."

**size**

public abstract int size

The pixel size of this Icon.

For example, 32 represents a 32x32 icon.

See Also:
"The size attribute of the Icon element of a Meta Type Resource."
Annotation Type ObjectClassDefinition

org.osgi.service.metatype.annotations

@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.TYPE)
public @interface ObjectClassDefinition

Generate a Meta Type Resource using the annotated type as an ObjectClassDefinition.

This annotation can be used without defining any element values as defaults can be generated from the annotated type. Each method of the annotated type has an implied AttributeDefinition annotation if not explicitly annotated.

This annotation is not processed at runtime. It must be processed by tools and used to generate a Meta Type Resource document for the bundle.

See Also:
“The OCD element of a Meta Type Resource.”

<table>
<thead>
<tr>
<th>Required Element Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String description</td>
<td>25</td>
</tr>
<tr>
<td>The human readable description of this ObjectClassDefinition.</td>
<td></td>
</tr>
<tr>
<td>String[] factoryPid</td>
<td>26</td>
</tr>
<tr>
<td>The factory PIDs associated with this ObjectClassDefinition.</td>
<td></td>
</tr>
<tr>
<td>Icons[] icon</td>
<td>26</td>
</tr>
<tr>
<td>The icon resources associated with this ObjectClassDefinition.</td>
<td></td>
</tr>
<tr>
<td>String id</td>
<td>24</td>
</tr>
<tr>
<td>The id of this ObjectClassDefinition.</td>
<td></td>
</tr>
<tr>
<td>String localization</td>
<td>25</td>
</tr>
<tr>
<td>The localization resource of this ObjectClassDefinition.</td>
<td></td>
</tr>
<tr>
<td>String name</td>
<td>25</td>
</tr>
<tr>
<td>The human readable name of this ObjectClassDefinition.</td>
<td></td>
</tr>
<tr>
<td>String[] pid</td>
<td>25</td>
</tr>
<tr>
<td>The PIDs associated with this ObjectClassDefinition.</td>
<td></td>
</tr>
</tbody>
</table>

Element Detail

id

public abstract String id

The id of this ObjectClassDefinition.

If not specified, the id of this ObjectClassDefinition is the fully qualified name of the annotated type using the dollar sign (’$’) to separate nested class names from the name of their enclosing class. The id is not to be confused with a PID which can be specified by the pid() or factoryPid() element.

Default: ”

See Also:
“The id attribute of the OCD element of a Meta Type Resource.”
name

public abstract String name

The human readable name of this ObjectClassDefinition.

If not specified, the name of this ObjectClassDefinition is derived from the \id(). For example, low line (’\_’ \u005F) and dollar sign (’\$’ \u0024) are replaced with space (’ ’ \u0020) and space is inserted between camel case words.

If the name begins with the percent sign (’\%’ \u0025), the name can be \localized.

Default: 

See Also:

"The name attribute of the OCD element of a Meta Type Resource."

description

public abstract String description

The human readable description of this ObjectClassDefinition.

If not specified, the description of this ObjectClassDefinition is the empty string.

If the description begins with the percent sign (’\%’ \u0025), the description can be \localized.

Default: 

See Also:

"The description attribute of the OCD element of a Meta Type Resource."

localization

public abstract String localization

The localization resource of this ObjectClassDefinition.

This refers to a resource property entry in the bundle that can be augmented with locale information. If not specified, the localization resource of this ObjectClassDefinition is the string "OSGI-INF/l10n/" followed by the fully qualified name of the annotated type.

Default:

See Also:

"The localization attribute of the OCD element of a Meta Type Resource."

pid

public abstract String[] pid

The PIDs associated with this ObjectClassDefinition.

For each specified PID, a Designate element with a pid attribute is generated that references this ObjectClassDefinition. If not specified, no Designate elements with pid attributes will be generated.

Default:

{}
**See Also:**

"The pid of the Designate element of a Meta Type Resource."

---

**factoryPid**

```java
public abstract String[] factoryPid
```

The factory PIDs associated with this ObjectClassDefinition.

For each specified factory PID, a Designate element with a factoryPid attribute is generated that references this ObjectClassDefinition. If not specified, no Designate elements with factoryPid attributes will be generated.

Default: `{}`

**See Also:**

"The factoryPid of the Designate element of a Meta Type Resource."

---

**icon**

```java
public abstract Icon[] icon
```

The icon resources associated with this ObjectClassDefinition.

For each specified Icon, an Icon element is generated for this ObjectClassDefinition. If not specified, no Icon elements will be generated.

Default: `{}`

**See Also:**

"The Icon element of a Meta Type Resource."
Annotation Type Option

org.osgi.service.metatype.annotations

@Retention(value=RetentionPolicy.CLASS)
@Target(value={})
public @interface Option

Option information for an AttributeDefinition.

See Also:
  AttributeDefinition.options()

<table>
<thead>
<tr>
<th>Required Element Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String label</td>
<td>27</td>
</tr>
<tr>
<td>String value</td>
<td>27</td>
</tr>
</tbody>
</table>

The human readable label of this Option.

The value of this Option.

Element Detail

label

public abstract String label

The human readable label of this Option.

If not specified, the label of this Option is the empty string.

If the label begins with the percent sign ('%' u0025), the label can be localized.

Default:

See Also:
  "The label attribute of the Option element of a Meta Type Resource."

value

public abstract String value

The value of this Option.

See Also:
  "The value attribute of the Option element of a Meta Type Resource."
7 Considered Alternatives

8 Security Considerations

The annotations do not have any security considerations.

9 Document Support

9.1 References


9.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>BJ Hargrave</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM</td>
</tr>
</tbody>
</table>

9.3 Acronyms and Abbreviations

9.4 End of Document
Abstract

This document defines the Java API that provides the information of network interfaces in an OSGi environment. The bundles can get not only information of network interfaces but notification when the configuration of network interfaces to be changed to use this API.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

<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 License</td>
<td>2</td>
</tr>
<tr>
<td>0.2 Trademarks</td>
<td>3</td>
</tr>
<tr>
<td>0.3 Feedback</td>
<td>3</td>
</tr>
<tr>
<td>0.4 Table of Contents</td>
<td>3</td>
</tr>
<tr>
<td>0.5 Terminology and Document Conventions</td>
<td>4</td>
</tr>
<tr>
<td>0.6 Revision History</td>
<td>4</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>7</td>
</tr>
<tr>
<td>2 Application Domain</td>
<td>7</td>
</tr>
<tr>
<td>3 Problem Description</td>
<td>8</td>
</tr>
<tr>
<td>4 Requirements</td>
<td>9</td>
</tr>
<tr>
<td>5 Technical Solution</td>
<td>10</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>10</td>
</tr>
<tr>
<td>5.2 Entities</td>
<td>10</td>
</tr>
<tr>
<td>5.3 NetworkAdapter Service</td>
<td>12</td>
</tr>
<tr>
<td>5.4 NetworkAddress Service</td>
<td>14</td>
</tr>
<tr>
<td>5.5 Network adapter type, IP address version and IP address scope</td>
<td>15</td>
</tr>
<tr>
<td>5.5.1 Network adapter type</td>
<td>15</td>
</tr>
<tr>
<td>5.5.2 IP address version and IP address scope</td>
<td>16</td>
</tr>
</tbody>
</table>
0.5 Terminology and Document Conventions

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

Source code is shown in this typeface.

0.6 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>Nov 18, 2013</td>
<td>Initial version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shigekuni Kondo, NTT Corporation, <a href="mailto:kondo.shigekuni@lab.ntt.co.jp">kondo.shigekuni@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>0.2</td>
<td>Feb 10, 2014</td>
<td>Based on the last meeting, the section 5 has changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Changed the design to service repository model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shigekuni Kondo, NTT Corporation, <a href="mailto:kondo.shigekuni@lab.ntt.co.jp">kondo.shigekuni@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>Revision</td>
<td>Date</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0.3</td>
<td>Feb 28, 2014</td>
<td>Based on the last meeting, the following points have been modified. rwif.displayname is changed to OPTIONAL. Interface name is changed. NwIfInfo --&gt; NetworkAdapter, NwIfNetAddress --&gt; NetworkAddress IPAdress Type is divided to IPAdresVersion and IPAdressScope. The functionality of configuration is removed. Shigekuni Kondo, NTT Corporation, <a href="mailto:kondo.shigekuni@lab.ntt.co.jp">kondo.shigekuni@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>0.4</td>
<td>Mar 6, 2014</td>
<td>Based on the last meeting, the following points have been modified. Update IPAdress Version and Scope. The order of registering (unregistering) NetworkAdapter and NetworkAddress is changed. Shigekuni Kondo, NTT Corporation, <a href="mailto:kondo.shigekuni@lab.ntt.co.jp">kondo.shigekuni@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>0.5</td>
<td>Mar 14, 2014</td>
<td>Based on the last conference call, the following points have been modified. Adding new service properties to NetworkAdapter service and NetworkAddress. Some service property names are modified. NetworkAadapterException class is removed. JavaDoc update. Some sentences are modified. Shigekuni Kondo, NTT Corporation, <a href="mailto:kondo.shigekuni@lab.ntt.co.jp">kondo.shigekuni@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>0.6</td>
<td>Mar 19, 2014</td>
<td>Fixed some sentences based on the comments from Evgeni. Added Chapter 8 Considered Alternatives section. Shigekuni Kondo, NTT Corporation, <a href="mailto:kondo.shigekuni@lab.ntt.co.jp">kondo.shigekuni@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>0.7</td>
<td>Apr 9, 2014</td>
<td>Fixed the packagename (Java doc). Added Chapter 9 Added Security section. Some sentences are modified. Shigekuni Kondo, NTT Corporation, <a href="mailto:kondo.shigekuni@lab.ntt.co.jp">kondo.shigekuni@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>Revision</td>
<td>Date</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 0.8      | Apr 30, 2014 | Based on the last CPEG conference call, the following points have been modified. Remove "service." prefix from service properties key.  
          |             | The constants “LAN” and “WAN” of Network Adapter Type are defined. The description of use case 2 and 3 is modified. All of service properties in NetwoekAdapter are changed to Required. Some sentences correction. |
| final    | Jul 7, 2014  | Finalized version                                                                                                                                                               |
|          |             | Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp                                                                                                                   |
1 Introduction

Java standard APIs (i.e. java.net.NetworkInterface, java.net.InetAddress) provide functions that allow IP network interface information, such as the IP address and MAC address to be obtained.

However, the bundle that wants to get network interface information has to monitor whether the information has changed or not for a certain period of time. Changes in network interface can be pushed to the bundles concerned, the need for polling by bundles can be eliminated.

In addition, some information cannot be obtained via Java standard APIs.

This RFC defines the Java API that provides the information of network interfaces in an OSGi environment. The bundles can get not only information of network interfaces but notification when the configuration of network interfaces to use this API.

2 Application Domain

There are many bundles that use the IP network to communicate with other networked devices. In particular, since a Residential Gateway (RGW) may have a number of network interfaces, each bundle running on the RGW needs to obtain an IP address and confirm whether the network interface associated with the allocated IP address suits the bundle’s requirements or not.

For example, a protocol adapter needs the IP address of a network interface on the wide area network side to communicate with an external server. UPnP device service bundle needs the IP address that can be used to communicate with devices in a local area network.

These bundles can acquire information about the network interface via the following Java standard APIs.

- java.net.NetworkInterface
- java.net.InetAddress
3 Problem Description

Many application bundles on the RGW provide services on IP networks. For example, a protocol adapter for DMT Admin Service, a http server established by HTTP Service bundle and UPnP device service bundle use IP networks. In those cases, the bundles need to get information about the network interface on the RGW such as IP address, MAC address, network interface name, and so on.

The information about the network interface can be obtained by using Java standard APIs which are java.net.NetworkInterface and java.net.InetAddress. However, these APIs fail to provide the features needed by the bundles when they use the IP network in the following situations:

[Problem 1] There is no feature that sends a notification when information of the network interface (i.e. IP address) changes during runtime, e.g. the connection status or the assigned IP address.

[Problem 2] There is no feature that can acquire the subnet mask of the network interface.

[Problem 3] Operating System specific bundles must be prepared because some information about network interface depends on the Operating System.

If these functions were available, it would be very useful for bundles that need to use the IP network. However, a standard API does not exist at this time, so it must be prepared for each environment.

3.1 Use Cases

Use case 1

The TR-069 protocol adapter bundle on a RGW needs to communicate with an Auto Configuration Server (ACS). The ACS needs to know the public IP address of the Residential Gateway to send a UDP packet to the protocol adapter bundle for a connection request. In this case, the bundle has to provide the IP address to the ACS when the bundle is started or the IP address has changed.

Use case 2

When an HTTP Service bundle is available, at least one HTTP server is expected to run. When the HTTP server needs to be assigned to a specific network interface, the HTTP Service bundle has to know the information of the network interface. The configurator bundle (For example, management agent) implemented the policy of the execution environment will detect the changes of IP address of the network interface and update the configuration of HTTP Service bundle.

Use case 3

The UPnP Device Service bundle needs to create the DatagramSocket for receiving and sending M-search messages. In the case of devices such as Residential Gateway, which has multi network interfaces, the UPnP bundle has to create a DatagramSocket that is bound to an appropriate local IP address. The configurator bundle (For example, management agent) implemented the policy of the
execution environment will detect the changes of IP address of the network interface and update the configuration of UPnP Device Service bundle.

**Use case 4**

An application bundle wants to obtain the subnet mask of the IP address to cover the situation in which the bundle needs to execute the Wake-up-On-LAN process.

**Use case 5**

An application wants to obtain information about available network services, such as available DNS Server, Log Server, NTP Server, or network characteristics, such as domain names, ARP cache timeouts, broadcast address, etc. For this, the local DHCP server can be queried to get those information.

**Use case 6**

A device running an OSGi framework in an mixed IPv4/IPv6 environment needs to get specific information about the network interface(s) in order to provide, for example, different services for the IPv4 and IPv6 environments.

## 4 Requirements

[REQ_1] The solution MUST provide means to send notifications to interested bundles whenever the information of network interface has changed. (i.e. The bundle is notified the information of IP address change from Network Interface Information Service implemented bundle)

[REQ_2] The solution MUST provide an API that can obtain information from a multiple network interfaces. Each network interface can provide information about multiple addresses. (An application bundle needs to know whether the network interface is a LAN interface or a WAN interface.).

[REQ_3] The solution MUST provide a mechanism that can provide the network interface information needed regardless of the Operating System type.

[REQ_4] The solution MUST provide the means of configuring network interface type. It will be defined for each environment (i.e. “LAN”, “WAN” that is bound to each logical interface).

[REQ_5] The solution MUST provide an API that can obtain the subnet mask of each IP address.

[REQ_6] The solution MUST support both IPv4 and IPv6 environments (mixed or separately) and the corresponding characteristics, for example IPv4 and IPv6 addresses, multi-prefixes, multicast etc.

[REQ_7] The solution SHOULD support the retrieval of MAC addresses for network interfaces.

[REQ_8] The solution MAY provide an API that allows alteration of network interface configurations.

[REQ_9] The solution MAY provide an API that can obtain the capability of network interface. (e.g. the physical type of network interface, list of BOOTP/DHCP command options, DNS server address, Default Gateway address, etc.)
5 Technical Solution

5.1 Introduction

When the IP address is changed, the bundles utilize the IP address information (i.e. Http Service bundle running HTTP Servers) is necessary to detect the fact of the change. When using a standard Java API, such as java.net.InetAddress and java.net.NetworkInterface, calls to confirm the IP address at regular intervals are required from the bundle itself. Since this is a process common to all bundles that need to detect any change in IP address information, provision of services to notify a change in IP address is very effective.

Therefore an API that provides a change notification feature for each piece of network interface information (including the IP address information) is investigated in this RFC document. In addition, this RFC defines APIs that provide the functionalities to obtain the network interface information and the information of IP address bound to the network interface.

The name of the network interface is dependent on the operating system. To allow the bundle implementation that uses the Network Interface Information Service is unaware of the differences in operating systems, a mechanism of identifying the network interface is necessary in a format that does not depend on the operating system. This is also defined in this RFC.

5.2 Entities

- Network Interface
  Available and activated network interfaces provided in the execution environment. In this specification, the unit of the network interface is the logical interface, not the physical interface.

- NetworkAdapter
  The OSGi service that provides information related to the Network Interface. This service provides functionalities corresponding to “java.net.NetworkInterface”.

- NetworkAddress
  The OSGi service that provides information of IP addresses available on the execution environment in which a Network Interface Information Service bundle is running.

- NetworkAdapterType
  An identifier of the network interface. It is independent of the operating system. The two type of identifier string is specified in this specification. This specification allows that Network Adapter type other than them can be defined by the platform provider in each environment. This identifier is used by user bundle to specify the network interface to be monitored.

- IPAddressVersion
  An identifier indicating the IP address version (i.e. Ipv4, IPv6). This identifier is defined in this specification. This identifier is used by a user bundle to specify the network interface to be monitored.
• **IPAdressScope**
  An identifier indicating the scope of IP address (i.e. GLOBAL, PRIVATE). This identifier is defined in this specification. This identifier is used by a user bundle to specify the network interface to be monitored.

![Class structure of Network Interface Information Service](image)

<Network Interface Information Bundle>

To register two kinds of services.

NetworkAdapter service provides network interface information, this bundle registers each logical interface as OSGi service.

NetworkAddress service provides each IP address information, this bundle registers each IP address as OSGi service.

NetworkAddress service is associated with specific NetworkAdapter service.

When information of network interface is changed, service properties of NetworkAdapter service and NetworkAddress service will be modified.

>User bundle>

Tracking necessary NetworkAdapter service and NetworkAddress service (using filter). This bundle is notified of a change in network interface information via Service Event.
5.3 NetworkAdapter Service

NetworkAdapter is an interface that provides information about single network interfaces provided by the execution environment. If multiple network interfaces are present, NetworkAdapter Services that correspond to each network interface must be registered.

NetworkAdapter service is registered with the service repository with service properties as shown in the following table.

Table 1. Service properties of NetworkAdapter Service

<table>
<thead>
<tr>
<th>The key of service property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>networkAdapter.type</td>
<td>Required property. Network interface type is set to a value.</td>
</tr>
<tr>
<td>networkAdapter.hardwareAddress</td>
<td>Required property. Hardware address (MAC address) is set to a value. This property can also be obtained from getHardwareAddress().</td>
</tr>
<tr>
<td>networkAdapter.name</td>
<td>Required property. Network interface name is set to a value. This property can also be obtained from getName().</td>
</tr>
<tr>
<td>networkAdapter.displayName</td>
<td>Required property. Network interface display name is set to a value. This property can also be obtained from getDisplayName().</td>
</tr>
<tr>
<td>Method</td>
<td>Adoption status</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>networkAdapter.isUp</td>
<td>Required property. The value is true when a network interface is up and running, otherwise it is false.</td>
</tr>
<tr>
<td>networkAdapter.isLoopback</td>
<td>Required property. The value is true when a network interface is a loopback interface, otherwise it is false.</td>
</tr>
<tr>
<td>networkAdapter.isPointToPoint</td>
<td>Required property. The value is true when a network interface is a point to point interface, otherwise it is false.</td>
</tr>
<tr>
<td>networkAdapter.isVirtual</td>
<td>Required property. The value is true when a network interface is a virtual interface, otherwise it is false.</td>
</tr>
<tr>
<td>networkAdapter.supportsMulticast</td>
<td>Required property. The value is true when a network interface supports multicasting, otherwise it is false.</td>
</tr>
<tr>
<td>networkAdapter.parent</td>
<td>Required property. Service PID of the NetworkAdapter service which is parent of this NetworkAdapter is specified.</td>
</tr>
<tr>
<td>networkAdapter.subInterface</td>
<td>Required property. Service PID of the NetworkAdapter service which is subinterface of this NetworkAdapter is specified.</td>
</tr>
</tbody>
</table>

When a network interface becomes available, NetworkAdapter service associated with the network interface is registered with the service repository. If the network interface becomes unavailable, the corresponding NetworkAdapter service is unregistered.

When the attribute values of the network interface are set to the service property changes, NetworkAdapter service is updated. NetworkAdapter interface provides a method corresponding to java.net.NetworkInterface in order to provide information on the associated network interface. However, this interface method does not support the Static method. In addition, because NetworkInterface object or InetAddress object is registered in the service repository as NetworkAdapter and NetworkAdress, the NetworkAdapter interface does not provide a method to get those objects. NetworkAdapter provides a method to retrieve the value of an attribute of a network interface.

### Table 2. Investigation of the method to be adopted based on Java standard API

<table>
<thead>
<tr>
<th>Method in java.net.NetworkInterface</th>
<th>Adoption status</th>
</tr>
</thead>
<tbody>
<tr>
<td>getByInetAddress(InetAddress)</td>
<td>Not adopted in this interface because NetworkAdapter service is registered.</td>
</tr>
<tr>
<td>getName(String)</td>
<td>Not adopted in this interface because NetworkAdapter service is registered.</td>
</tr>
<tr>
<td>getDisplayName()</td>
<td>Adopted in this interface.</td>
</tr>
<tr>
<td>getHardwareAddress()</td>
<td>Adopted in this interface.</td>
</tr>
<tr>
<td>getInetAddresses()</td>
<td>Not adopted in this interface because InetAddress object is provided</td>
</tr>
</tbody>
</table>
5.4 NetworkAddress Service

NetworkAddress interface provides information of IP addresses available in which execution environment on a Network Interface Information Service bundle is running.

NetworkAddress service is registered with the service repository together with service properties as shown in the following table.

<table>
<thead>
<tr>
<th>The key of service property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>networkAdapter.type</td>
<td>Required property. Network interface type is set to a value.</td>
</tr>
<tr>
<td>ipAddress.version</td>
<td>Required property. IP address version is set to a value.</td>
</tr>
<tr>
<td>ipAddress.scope</td>
<td>Required property. IP address scope is set to a value.</td>
</tr>
<tr>
<td>ipAddress</td>
<td>Required property. IP address String is set to a value.</td>
</tr>
</tbody>
</table>
subnetmask.length | Required property. subnet mask length of the required properties IPv4, or IPv6 prefix length is set to a value.

networkAdapter.pid | Required property. Service ID of the NetworkAdapter service corresponding to the network interface binding this IP address is set to a value.

NetworkAddress service is registered with the service repository for each available IP address.

When associated IP addresses are deleted, or the network interface to which the IP address is bound becomes unavailable, the NetworkAddress service is unregistered. When the associated IP address changes, NetworkAddress service is updated. The user bundle can detect the change of IP address by monitoring the registration or unregistering, updating of NetworkAddress service.

Because IP addresses are bound to the network interface, if any, Service PID of the associated NetworkAdapter service and its network interface type are set to service property.

NetworkAdapter service MUST be registered after the all associated NetworkAddress services are registered. On the other hand, when unregistering services, after associated NetworkAdapter service is unregistered, NetworkAddress of all related services are unregistered.

5.5 Network adapter type, IP address version and IP address scope

5.5.1 Network adapter type

Identifying the network interface is possible by using the network interface name.

However, since the network interface name is an identifier that is dependent on the operating system, if network interface name is used as identifier, user bundles must be implemented to be aware of the operating system. Therefore, in this specification, “network interface type” which is independent of the operating system, is used to identify the network interface. The network interface type string of “LAN” and “WAN” are defined in this specification. This specification allows that Network Adapter type other than “LAN” and “WAN” can be defined by the platform provider in each environment. It may be provided by the platform provider on which Network Interface Information Service bundle is running. Network interface type “LAN” indicates the network interface connects to a local area network. Network interface type “WAN” indicates the network interface connects to an external network (i.e. Internet). If a bundle wants to obtain the information of the network interface connected to the Internet, the bundle is able to get it by obtaining NetworkAdapter service which sets "SERVICE_NETWORKADAPTER_TYPE = WAN" to service property from the service repository.

Table 4. Network Adapter Type
### Network Interface Type

<table>
<thead>
<tr>
<th>Network Interface Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAN</td>
<td>The network interface to connect to a local area network.</td>
</tr>
<tr>
<td>WAN</td>
<td>The network interface to connect to an external network (i.e. Internet).</td>
</tr>
</tbody>
</table>

#### 5.5.2 IP address version and IP address scope

This specification defines “IP address version” and “IP address scope” as IP address version and IP address scope to be narrowed down the IP address by user bundle as follows.

**Table 5. IP Address Version**

<table>
<thead>
<tr>
<th>IP Address Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPV4</td>
<td>IP address version which means IPv4 address.</td>
</tr>
<tr>
<td>IPV6</td>
<td>IP address version which means IPv6 address.</td>
</tr>
</tbody>
</table>

**Table 6. IP Address Scope (T.B.D)**

<table>
<thead>
<tr>
<th>IP Address Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL</td>
<td>IP address scope which means global address.</td>
</tr>
<tr>
<td>PRIVATE_USE</td>
<td>IP address scope which means private address.</td>
</tr>
<tr>
<td>LOOPBACK</td>
<td>IP address scope which means loopback address.</td>
</tr>
<tr>
<td>LINKLOCAL</td>
<td>IP address scope which means linklocal address.</td>
</tr>
<tr>
<td>UNIQUE_LOCAL</td>
<td>IP address scope which means unique-local address.</td>
</tr>
<tr>
<td>UNSPECIFIED</td>
<td>IP address scope which means the absence of an address.</td>
</tr>
</tbody>
</table>

If a bundle which wants to check for an IP address of the IPv4 global, the bundle is able to confirm by obtaining NetworkAddress service which sets "SERVICE_IPADDRESS_VERSION = IPV4" and "SERVICE_IPADDRESS_SCOPE = GLOBAL" to service property from service repository.
6 Data Transfer Objects

This RFC will not provide Data Transfer Objects.

7 Javadoc
## Package Summary

<table>
<thead>
<tr>
<th>Package</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.service.networkadapter</td>
<td>19</td>
</tr>
</tbody>
</table>
## Package org.osgi.service.networkadapter

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NetworkAdapter</strong></td>
<td>20</td>
</tr>
<tr>
<td>NetworkAdapter is an interface that provides information about single network interfaces provided by the execution environment.</td>
<td></td>
</tr>
<tr>
<td><strong>NetworkAddress</strong></td>
<td>27</td>
</tr>
<tr>
<td>This interface represents an IP address information.</td>
<td></td>
</tr>
</tbody>
</table>
public interface NetworkAdapter

NetworkAdapter is an interface that provides information about single network interfaces provided by the execution environment.

If multiple network interfaces are present, NetworkAdapter Services that correspond to each network interface must be registered. Network interface information service is set the following information as service property.

1. NETWORKADAPTER_TYPE: Network Adapter Type
2. NETWORKADAPTER_DISPLAYNAME: Network Interface Display Name
3. NETWORKADAPTER_NAME: Network Interface Name
4. NETWORKADAPTER_HARDWAREADDRESS: Hardware Address
5. NETWORKADAPTER_IS_UP: Running status of Network Interface
6. NETWORKADAPTER_IS_LOOPBACK: To check loopback interface
7. NETWORKADAPTER_IS_POINTTOPOINT: To check point to point interface
8. NETWORKADAPTER_IS_VIRTUAL: To check virtual interface
9. NETWORKADAPTER_SUPPORTS_MULTICAST: To check supports multicasting
10. NETWORKADAPTER_PARENT: The PID of parent Network Interface
11. NETWORKADAPTER_SUBINTERFACE: The PID of sub Network Interface

When a network interface becomes available, NetworkAdapter service associated with the network interface is registered with the service repository. If the network interface becomes unavailable, the corresponding NetworkAdapter service is unregistered.

When the attribute values of the network interface are set to the service property changes, NetworkAdapter service is updated. NetworkAdapter interface provides a method corresponding to java.net.NetworkInterface in order to provide information on the associated network interface. However, this interface method does not support the Static method. In addition, because NetworkInterface object or InetAddress object is registered in the service repository as NetworkAdapter and NetworkAdress, the NetworkAdapter interface does not provide a method to get those objects. NetworkAdapter provides a method to retrieve the value of an attribute of a network interface.

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String LAN</td>
<td>23</td>
</tr>
<tr>
<td>String NETWORKADAPTER_DISPLAYNAME</td>
<td>21</td>
</tr>
<tr>
<td>String NETWORKADAPTER_HARDWAREADDRESS</td>
<td>22</td>
</tr>
<tr>
<td>String NETWORKADAPTER_IS_LOOPBACK</td>
<td>22</td>
</tr>
<tr>
<td>String NETWORKADAPTER_IS_POINTTOPOINT</td>
<td>22</td>
</tr>
<tr>
<td>String NETWORKADAPTER_IS_UP</td>
<td>22</td>
</tr>
<tr>
<td>String NETWORKADAPTER_IS_VIRTUAL</td>
<td>22</td>
</tr>
<tr>
<td>String NETWORKADAPTER_NAME</td>
<td>22</td>
</tr>
</tbody>
</table>

The string of networkadapter type which means the network interface to connect to a local area network.

The key string of "networkAdapter.displayName" service property.

The key string of "networkAdapter.hardwareAddress" service property.

The key string of "networkAdapter.isLoopback" service property.

The key string of "networkAdapter.isPointToPoint" service property.

The key string of "networkAdapter.isUp" service property.

The key string of "networkAdapter.isVirtual" service property.

The key string of "networkAdapter.name" service property.
Interface NwIfInetAddress

String NETWORKADAPTER_PARENT
   The key string of "networkAdapter.parent" service property.

String NETWORKADAPTER_SUBINTERFACE
   The key string of "networkAdapter.subInterface" service property.

String NETWORKADAPTER_SUPPORTS_MULTICAST
   The key string of "networkAdapter.supportsMulticast" service property.

String NETWORKADAPTER_TYPE
   The key string of "networkAdapter.type" service property.

String WAN
   The string of networkadapter type which means the network interface to connect to an external network (i.e.

Method Summary

<table>
<thead>
<tr>
<th>String</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>getDisplayName()</td>
<td>Returns the network interface display name of &quot;networkAdapter.displayName&quot; service property value.</td>
</tr>
<tr>
<td>byte[]</td>
<td>getHardwareAddress()</td>
<td>Returns the MAC address of &quot;networkAdapter.hardwareAddress&quot; service property value.</td>
</tr>
<tr>
<td>int</td>
<td>getMTU()</td>
<td>Returns the Maximum Transmission Unit (MTU) of this interface.</td>
</tr>
<tr>
<td>String</td>
<td>getName()</td>
<td>Returns the network interface name of &quot;networkAdapter.name&quot; service property value.</td>
</tr>
<tr>
<td>String</td>
<td>getNetworkAdapterType()</td>
<td>Returns the network interface type of &quot;networkAdapter.type&quot; service property value.</td>
</tr>
<tr>
<td>boolean</td>
<td>isLoopback()</td>
<td>Returns whether a network interface is a loopback interface.</td>
</tr>
<tr>
<td>boolean</td>
<td>isPointToPoint()</td>
<td>Returns whether a network interface is a point to point interface.</td>
</tr>
<tr>
<td>boolean</td>
<td>isUp()</td>
<td>Returns whether a network interface is up and running.</td>
</tr>
<tr>
<td>boolean</td>
<td>isVirtual()</td>
<td>Returns whether this interface is a virtual interface (also called subinterface).</td>
</tr>
<tr>
<td>boolean</td>
<td>supportsMulticast()</td>
<td>Returns whether a network interface supports multicasting or not.</td>
</tr>
</tbody>
</table>

Field Detail

NETWORKADAPTER_TYPE

public static final String NETWORKADAPTER_TYPE = "networkAdapter.type"

   The key string of "networkAdapter.type" service property is specified.

NETWORKADAPTER_DISPLAYNAME

public static final String NETWORKADAPTER_DISPLAYNAME = "networkAdapter.displayName"
Interface NwIfNetAddress

The key string of "networkAdapter.displayName" service property.

Network Interface Display Name is specified.

NETWORKADAPTER_NAME

public static final String NETWORKADAPTER_NAME = "networkAdapter.name"

The key string of "networkAdapter.name" service property.

Network Interface Name is specified.

NETWORKADAPTER_HARDWAREADDRESS

public static final String NETWORKADAPTER_HARDWAREADDRESS = "networkAdapter.hardwareAddress"

The key string of "networkAdapter.hardwareAddress" service property.

Hardware Address is specified.

NETWORKADAPTER_IS_UP

public static final String NETWORKADAPTER_IS_UP = "networkAdapter.isUp"

The key string of "networkAdapter.isUp" service property.

The value is true when a network interface is up and running, otherwise it is false.

NETWORKADAPTER_IS_LOOPBACK

public static final String NETWORKADAPTER_IS_LOOPBACK = "networkAdapter.isLoopback"

The key string of "networkAdapter.isLoopback" service property.

The value is true when a network interface is a loopback interface, otherwise it is false.

NETWORKADAPTER_IS_POINTTOPOINT

public static final String NETWORKADAPTER_IS_POINTTOPOINT = "networkAdapter.isPointToPoint"

The key string of "networkAdapter.isPointToPoint" service property.

The value is true when a network interface is a point to point interface, otherwise it is false.

NETWORKADAPTER_IS_VIRTUAL

public static final String NETWORKADAPTER_IS_VIRTUAL = "networkAdapter.isVirtual"
**Interface NwIfInetAddress**

The key string of "networkAdapter.isVirtual" service property. The value is true when a network interface is a virtual interface, otherwise it is false.

### NETWORKADAPTER_SUPPORTS_MULTICAST

```java
public static final String NETWORKADAPTER_SUPPORTS_MULTICAST = "networkAdapter.supportsMulticast"
```

The key string of "networkAdapter.supportsMulticast" service property. The value is true when a network interface supports multicasting, otherwise it is false.

### NETWORKADAPTER_PARENT

```java
public static final String NETWORKADAPTER_PARENT = "networkAdapter.parent"
```

The key string of "networkAdapter.parent" service property. Service PID of the NetworkAdapter service which is parent of this NetworkAdapter is specified.

### NETWORKADAPTER_SUBINTERFACE

```java
public static final String NETWORKADAPTER_SUBINTERFACE = "networkAdapter.subInterface"
```

The key string of "networkAdapter.subInterface" service property. Service PID of the NetworkAdapter service which is subinterface of this NetworkAdapter is specified.

### WAN

```java
public static final String WAN = "WAN"
```

The string of networkadapter type which means the network interface to connect to an external network (i.e. Internet).

### LAN

```java
public static final String LAN = "LAN"
```

The string of networkadapter type which means the network interface to connect to a local area network.

## Method Detail

### getNetworkAdapterType

```java
String getNetworkAdapterType() throws RuntimeException
```

OSGi Javadoc -- 13/11/18
Returns the network interface type of "networkAdapter.type" service property value.

**Returns:**
Network Interface Type

### getDisplayName

String `getDisplayName()`

Returns the network interface display name of "networkAdapter.displayname" service property value.

**Returns:**
Network Interface Display Name

### getName

String `getName()`

Returns the network interface name of "networkAdapter.name" service property value.

**Returns:**
Network Interface Name

### getHardwareAddress

byte[] `getHardwareAddress()`

Returns the MAC address of "networkAdapter.hardwareAddress" service property value.

**Returns:**
Hardware Address

### getMTU

int `getMTU()`

Throws `SocketException`

Returns the Maximum Transmission Unit (MTU) of this interface.

**Returns:**
The value of the MTU for that interface.

**Throws:**
`SocketException` - If an I/O error occurs.
**isLoopback**

```java
boolean isLoopback()
    throws SocketException
```

Returns whether a network interface is a loopback interface.

Returns:
true if the interface is a loopback interface.

Throws:
SocketException - If an I/O error occurs.

**isPointToPoint**

```java
boolean isPointToPoint()
    throws SocketException
```

Returns whether a network interface is a point to point interface.

Returns:
true if the interface is a point to point interface.

Throws:
SocketException - If an I/O error occurs.

**isUp**

```java
boolean isUp()
    throws SocketException
```

Returns whether a network interface is up and running.

Returns:
true if the interface is up and running.

Throws:
SocketException - If an I/O error occurs.

**isVirtual**

```java
boolean isVirtual()
```

Returns whether this interface is a virtual interface (also called subinterface). Virtual interfaces are, on some systems, interfaces created as a child of a physical interface and given different settings (like address or MTU). Usually the name of the interface will the name of the parent followed by a colon (:) and a number identifying the child since there can be several virtual interfaces attached to a single physical interface.
**Interface NwIfInetAddress**

**Returns:**
true if this interface is a virtual interface.

---

### supportsMulticast

```java
boolean supportsMulticast() throws SocketException
```

**Returns:**
whether a network interface supports multicasting or not.

**Returns:**
true if the interface supports Multicasting.

**Throws:**
SocketException - If an I/O error occurs.
public interface NetworkAddress

This interface represents an IP address information.

NetworkAddress interface provides information of IP addresses available in which execution environment on a Network Interface Information Service bundle is running. IP address information service is set the following information as service property.

1. NETWORKADAPTER_TYPE : Network Interface Type
2. IPADDRESS_VERSION : IP Address Version
3. IPADDRESS_SCOPE : IP Address Scope
4. IPADDRESS : IP Address
5. SUBNETMASK_LENGTH : Subnet Mask Length(IPv4) or Prefix Length(IPv6)
6. NETWORKADAPTER_PID : Service PID of the NetworkAdapter service to which this service belongs

NetworkAddress service is registered with the service repository for each available IP address. When associated IP addresses are deleted, or the network interface to which the IP address is bound becomes unavailable, the NetworkAddress service is unregistered. When the associated IP address changes, NetworkAddress service is updated. The user bundle can detect the change of IP address by monitoring the registration or unregistering, updating of NetworkAddress service. Because IP addresses are bound to the network interface, if any, Service PID of the associated NetworkAdapter service and its network interface type are set to service property. NetworkAdapter service MUST be registered after the all associated NetworkAddress services are registered. On the other hand, when unregistering services, after associated NetworkAdapter service is unregistered, NetworkAddress of all related services are unregistered.

Field Summary

<table>
<thead>
<tr>
<th>String</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPADDRESS</td>
<td>29</td>
</tr>
<tr>
<td>The key string of &quot;ipAddress&quot; service property.</td>
<td></td>
</tr>
<tr>
<td>IPADDRESS_SCOPE</td>
<td>28</td>
</tr>
<tr>
<td>The key string of &quot;IpAddress.scope&quot; service property.</td>
<td></td>
</tr>
<tr>
<td>IPADDRESS_SCOPE_GLOBAL</td>
<td>29</td>
</tr>
<tr>
<td>The string of IP address scope which means global address.</td>
<td></td>
</tr>
<tr>
<td>IPADDRESS_SCOPE.getHost</td>
<td>30</td>
</tr>
<tr>
<td>The string of IP address scope which means &quot;This host on this network&quot;.</td>
<td></td>
</tr>
<tr>
<td>IPADDRESS_SCOPE_LINKED_SCOPED_UNICAST</td>
<td>31</td>
</tr>
<tr>
<td>The string of IP address scope which means &quot;Linked-Scoped Unicast&quot;.</td>
<td></td>
</tr>
<tr>
<td>IPADDRESS_SCOPE_LINKLOCAL</td>
<td>30</td>
</tr>
<tr>
<td>The string of IP address scope which means &quot;Link Local&quot;.</td>
<td></td>
</tr>
<tr>
<td>IPADDRESS_SCOPE_LOOPBACK</td>
<td>30</td>
</tr>
<tr>
<td>The string of IP address scope which means &quot;Loopback&quot;.</td>
<td></td>
</tr>
<tr>
<td>IPADDRESS_SCOPE_PRIVATE_USE</td>
<td>30</td>
</tr>
<tr>
<td>The string of IP address scope which means &quot;Private-Use Networks&quot;.</td>
<td></td>
</tr>
<tr>
<td>IPADDRESS_SCOPE_SHARED</td>
<td>31</td>
</tr>
<tr>
<td>The string of IP address scope which means &quot;Shared Address Space&quot;.</td>
<td></td>
</tr>
<tr>
<td>IPADDRESS_SCOPE_UNIQUE_LOCAL</td>
<td>30</td>
</tr>
<tr>
<td>The string of IP address scope which means &quot;Unique-Local&quot;.</td>
<td></td>
</tr>
<tr>
<td>IPADDRESS_SCOPE_UNSPECIFIED</td>
<td>30</td>
</tr>
<tr>
<td>The string of IP address scope which means &quot;Unspecified Address&quot;.</td>
<td></td>
</tr>
</tbody>
</table>
## Interface NwIfInetAddressList

<table>
<thead>
<tr>
<th>String</th>
<th>IPADDRESS_VERSION</th>
<th>The key string of &quot;ipAddress.version&quot; service property.</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>IPADDRESS_VERSION_4</td>
<td>The string of IP address version which means IP address version 4.</td>
</tr>
<tr>
<td>String</td>
<td>IPADDRESS_VERSION_6</td>
<td>The string of IP address version which means IP address version 6.</td>
</tr>
<tr>
<td>String</td>
<td>NETWORKADAPTER_PID</td>
<td>The key string of &quot;networkAdapter.id&quot; service property.</td>
</tr>
<tr>
<td>String</td>
<td>NETWORKADAPTER_TYPE</td>
<td>The key string of &quot;networkAdapter.type&quot; service property.</td>
</tr>
<tr>
<td>String</td>
<td>SUBNETMASK_LENGTH</td>
<td>The key string of &quot;subnetmask.length&quot; service property.</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getInetAddress()</td>
<td>Returns the InetAddress object of this IP address.</td>
<td>32</td>
</tr>
<tr>
<td>String getIpAddress()</td>
<td>Returns the IP address of &quot;ipaddress&quot; service property value.</td>
<td>31</td>
</tr>
<tr>
<td>String getIpAddressScope()</td>
<td>Returns the IP address scope of &quot;ipaddress.scope&quot; service property value.</td>
<td>31</td>
</tr>
<tr>
<td>String getIpAddressVersion()</td>
<td>Returns the IP address version of &quot;ipaddress.version&quot; service property value.</td>
<td>31</td>
</tr>
<tr>
<td>String getNetworkAdapterPid()</td>
<td>Returns the &quot;networkadapter.pid&quot; service property value.</td>
<td>32</td>
</tr>
<tr>
<td>String getNetworkAdapterType()</td>
<td>Returns the network interface type of &quot;networkAdapter.type&quot; service property value.</td>
<td>31</td>
</tr>
<tr>
<td>int getSubnetMaskLength()</td>
<td>Returns the &quot;subnetmask.length&quot; service property value.</td>
<td>32</td>
</tr>
</tbody>
</table>

### Field Detail

**NETWORKADAPTER_TYPE**

```java
public static final String NETWORKADAPTER_TYPE = "networkAdapter.type"
```

The key string of "networkAdapter.type" service property. Network Interface Type is specified.

**IPADDRESS_VERSION**

```java
public static final String IPADDRESS_VERSION = "ipAddress.version"
```

The key string of "ipAddress.version" service property. IP Address Type is specified.

**IPADDRESS_SCOPE**

```java
public static final String IPADDRESS_SCOPE = "ipAddress.scope"
```

OSGi Javadoc -- 13/11/18
Interface NwIfNetAddressList

The key string of "ipAddress.scope" service property.
IP Address Type is specified.

IPADDRESS

public static final String IPADDRESS = "ipAddress"

The key string of "ipAddress" service property.
IP Address is specified.

SUBNETMASK_LENGTH

public static final String SUBNETMASK_LENGTH = "subnetmask.length"

The key string of "subnetmask.length" service property.
Subnet Mask Length(IPv4) or Prefix Length(IPv6) is specified.

NETWORKADAPTER_PID

public static final String NETWORKADAPTER_PID = "networkAdapter.pid"

The key string of "networkAdapter.id" service property.
Service PID of the interface information service to which it belongs is specified.

IPADDRESS_VERSION_4

public static final String IPADDRESS_VERSION_4 = "IPV4"

The string of IP address version which means IP address version 4.

IPADDRESS_VERSION_6

public static final String IPADDRESS_VERSION_6 = "IPV6"

The string of IP address version which means IP address version 6.

IPADDRESS_SCOPE_GLOBAL

public static final String IPADDRESS_SCOPE_GLOBAL = "GLOBAL"

The string of IP address scope which means global address.
The global address is defined as the address other than the address defined in the RFC6890.
IPADDRESS_SCOPE_PRIVATE_USE

public static final String IPADDRESS_SCOPE_PRIVATE_USE = "PRIVATE_USE"

The string of IP address scope which means "Private-Use Networks". See RFC6890 for the definition of "Private-Use Networks".

IPADDRESS_SCOPE_LOOPBACK

public static final String IPADDRESS_SCOPE_LOOPBACK = "LOOPBACK"

The string of IP address scope which means "Loopback". See RFC6890 for the definition of "Loopback".

IPADDRESS_SCOPE_LINKLOCAL

public static final String IPADDRESS_SCOPE_LINKLOCAL = "LINKLOCAL"

The string of IP address scope which means "Link Local". See RFC6890 for the definition of "Link Local".

IPADDRESS_SCOPE_UNIQUE_LOCAL

public static final String IPADDRESS_SCOPE_UNIQUE_LOCAL = "UNIQUE_LOCAL"

The string of IP address scope which means "Unique-Local". See RFC6890 for the definition of "Unique-Local".

IPADDRESS_SCOPE_UNSPECIFIED

public static final String IPADDRESS_SCOPE_UNSPECIFIED = "UNSPECIFIED"

The string of IP address scope which means "Unspecified Address". See RFC6890 for the definition of "Unspecified Address".

IPADDRESS_SCOPE_HOST

public static final String IPADDRESS_SCOPE_HOST = "HOST"

The string of IP address scope which means "This host on this network". See RFC6890 for the definition of "This host on this network".
IPADDRESS_SCOPE_SHARED

public static final String IPADDRESS_SCOPE_SHARED = "SHARED"

The string of IP address scope which means "Shared Address Space". See RFC6890 for the definition of "Shared Address Space".

IPADDRESS_SCOPE_LINKED_SCOPED_UNICAST

public static final String IPADDRESS_SCOPE_LINKED_SCOPED_UNICAST = "LINKED_SCOPED_UNICAST"

The string of IP address scope which means "Linked Scoped Unicast". See RFC6890 for the definition of "Linked Scoped Unicast".

Method Detail

getNetworkAdapterType

String getNetworkAdapterType()

Returns the network interface type of "networkAdapter.type" service property value.

Returns:
   Network Interface Type

getAddressVersion

String getIpAddressVersion()

Returns the IP address version of "ipaddress.version" service property value.

Returns:
   IP Address Version

getAddressScope

String getIpAddressScope()  

Returns the IP address scope of "ipaddress.scope" service property value.

Returns:
   IP Address Scope

getAddress

String getIpAddress()
Interface NwIfInetAddressList

Returns the IP address of "ipaddress" service property value.

Returns:
   IP Address string

getInetAddress

InetAddress getInetAddress()

Returns the InetAddress object of this IP address.
Returned object is created from "ipaddress" service property value.

Returns:
   InetAddress

getSubnetMaskLength

int getSubnetMaskLength()

Returns the "subnetmask.length" service property value.

Returns:
   Subnet Mask Length(IPv4) or Prefix Length(IPv6)

getNetworkAdapterPid

String getNetworkAdapterPid()

Returns the "networkadapter.pid" service property value.

Returns:
   Service ID of the interface information service to which it belongs
8 Considered Alternatives

8.1 Whiteboard pattern model

<NwInfo Bundle>

Provides (exports) Listener service interface, and gets the Listener services provided from user bundle. When information of a network interface is changed, this bundle prepares list of network interface information and calls back the Listener services. This bundle will provide a filter mechanism. User bundle can get only the information necessary to use the functionality.

<User bundle>

Registers a Listener service, and waits for notification of network interface information.

Fig. 3 Overview of Network Interface Information Service

(Whiteboard pattern model)
9 Security Considerations

The user bundles that want to know information of one or more Network Interfaces should be assigned ServicePermission[NetworkAdapter, GET] and ServicePermission[NetworkAddress, GET].

Filter Based Permissions can also be utilized for assigning ServicePermission. If the platform provider wants to control a bundle's access to the service, the following example of ServicePermission can be set.

ServicePermission["&(objectClass=org.osgi.service.networkadapter.NetworkAdapter)(networkAdapter.type=LAN)", GET]

ServicePermission["&(objectClass=org.osgi.service.networkadapter.NetworkAddress)(networkAdapter.type=LAN)(ipAddress.version=IPV4)(ipAddress.scope=PRIVATE_USE)", GET]

10 Document Support

10.1 References


10.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Shigekuni KONDO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>NTT Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>1-1, Hikari-no-oka, Yokosuka-shi, 238-0847, Kanagawa, Japan</td>
</tr>
<tr>
<td>Voice</td>
<td>+81 46 859 3444</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:kondo.shigekuni@lab.ntt.co.jp">kondo.shigekuni@lab.ntt.co.jp</a></td>
</tr>
</tbody>
</table>
10.3 Acronyms and Abbreviations

10.4 End of Document
Device Abstraction Layer Functions

Final

65 Pages

Abstract

Defines a core set of functions to RFC-0196 Device Abstraction Layer. They provide an interoperability between the different specific domains.
0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0 Document Information

0.1 License .................................................................2
0.2 Trademarks .............................................................3
0.3 Feedback ...............................................................3
0.4 Table of Contents .....................................................3
0.5 Terminology and Document Conventions .....................4
0.6 Revision History .......................................................4

1 Introduction ..........................................................4

2 Application Domain ...................................................5

3 Problem Description ..................................................5

4 Requirements ................................................................6

5 Technical Solution ......................................................7

5.1 Core Functions .......................................................7
5.1.1 BooleanControl Function .......................................7
5.1.2 BooleanSensor Function .........................................8
5.1.3 MultiLevelControl Function ....................................8
5.1.4 MultiLevelSensor Function .....................................9
5.1.5 Meter Function .....................................................9
5.1.6 Alarm Function ....................................................10
5.1.7 Keypad Function ..................................................10
5.1.8 WakeUp Function ..................................................11

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.
1 Introduction

OSGi is gaining popularity as enabling technology for building embedded system in residential and M2M markets. In these contexts it is often necessary to communicate with IP and non-IP devices by using various protocols such as ZigBee, Z-Wave, KNX, UPnP etc. In order to provide a convenient programming model suitable for the realization of end-to-end services it is very useful to define and apply an abstraction layer which unifies the work with devices supporting different protocols.

This RFC defines a core set of functions to RFC-0196 Device Abstraction Layer. They provide an interoperability between the different specific domains. The set can be reused, extended or fully replaced into specific domain.
2 Application Domain

OSGi Device Abstraction Layer RFC-0196 don't define functions, but their common representation. It doesn't guarantee interoperability between the different domains. In this way, the same functionality can be modeled in different ways and the applications will be bind to the specific interfaces.

For example, a meter function can be defined in the smart home domain as an energy meter, but in the security domain as a time meter (timer).

In order to unify the application access to the basic functionalities, a core of functions is required.

3 Problem Description

The applications need to know about a set of OSGi Device Abstraction Layer functions to operate with them. They can execute operations, set and receive property values.

Illustration 1 shows one possible approach for working with heterogeneous functions, which are related to the same functionality. The smart home meter and vendor specific meter are all about the same kind of information. They collect metering information.

In this case each application must use specific API for this function. One obvious disadvantage of this model is that when a new function type is added the applications must be modified in order to support it.

Much better is the approach from Illustration 2 which is defined by this RFC.
In this case a core functions are introduced to ensure the interoperability between the different domains. Thus the following advantages are achieved:

- The application programmers can work with the same set of functions.
- The functions can be reused or extended.
- The application can work without modifications when a new vendor function types are registered.

### 4 Requirements

**Requirement 1.** The solution MUST define API for controlling devices which is applicable for all relevant device protocols.

**Requirement 2.** The solution MUST define API for controlling devices which is independent from the device protocols.

**Requirement 3.** The solution MUST include device access control based on user and application permissions compliant with the OSGi security model.

**Requirement 4.** The solution SHOULD be mappable to other relevant standards such as HGI, ETSI M2M and BBF handling the remote access to device networks.

**Requirement 5.** The solution MUST be applicable to the changeable device behavior. Sleeping/power saving devices can go and stay offline for a long time, but should be available in the defined API.
5 Technical Solution

5.1 Core Functions

Concrete function interfaces have to be defined to unify the access and control of the basic operations and related properties. The current section specifies the minimal basic set of such functionality. It can be reused and extended to cover more specific scenarios. They are about the control, monitoring and metering information.

5.1.1 BooleanControl Function

BooleanControl function provides a binary control support. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later. The full function definition is available in the next table.

<table>
<thead>
<tr>
<th>BooleanControl</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td><strong>Operations</strong></td>
</tr>
<tr>
<td>reverse</td>
</tr>
<tr>
<td>setTrue</td>
</tr>
<tr>
<td>setFalse</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
</tr>
<tr>
<td>data</td>
</tr>
<tr>
<td><strong>Types</strong></td>
</tr>
<tr>
<td>light, door, window, power, other type defined in org.osgi.service.dal.functions.Types or vendor specific type.</td>
</tr>
</tbody>
</table>

BooleanData data structure is used to provide information about the function state. That data object contains the boolean value, the value collecting time and additional metadata. The immutable BooleanData.value field is accessible with BooleanData.getValue() getter.

The function class diagram is depicted on Illustration 3. The next code snippet sets to true all BooleanControl functions.

```java
final ServiceReference[] booleanControlSRefs = context.getServiceReferences(
        BooleanControl.class.getName(), null);
if (null == booleanControlSRefs) {
    return; // no such services
}
for (int i = 0; i < booleanControlSRefs.length; i++) {
    Copyright © OSGi Alliance 2014 All Rights Reserved
```
final BooleanControl booleanControl = (BooleanControl) context.getService(
    binaryControlSRefs[i]);
if (null != booleanControl) {
    booleanControl.setTrue();
}

5.1.2 BooleanSensor Function

BooleanSensor function provides binary sensor monitoring. It reports its state when an important event is available. There are no operations. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later. The full function definition is available in the next table.

<table>
<thead>
<tr>
<th>BooleanSensor</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
<td>Contains the current state of BooleanSensor. The property access can be: readable and eventable.</td>
</tr>
<tr>
<td>Types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light, gas, smoke, door, window, power, rain, contact, fire, occupancy, water, motion, other type defined in org.osgi.service.dal.functions.Types or vendor specific type.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BooleanSensor and BooleanControl are using the same BooleanData data structure to provide information about the state. For more details see the definition in BooleanControl Function. The function class diagram is depicted on Illustration 3.

5.1.3 MultiLevelControl Function

MultiLevelControl function provides multi-level control support. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later. The full function definition is available in the next table.

<table>
<thead>
<tr>
<th>MultiLevelControl</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
<td>Contains the current state of MultiLevelControl. The property access can be: readable, writable and eventable.</td>
</tr>
<tr>
<td>Types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light, temperature, flow, pressure, humidity, gas, smoke, door, window, liquid, power, noisiness, other type defined in org.osgi.service.dal.functions.Types or vendor specific type.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LevelData data structure is used to provide information about the function level. That data object contains the BigDecimal value and the value unit. The measurement unit is used as it’s defined in RFC-0196 OSGi Device Abstraction Later. The immutable LevelData.unit field is accessible with LevelData.getUnit() getter. The immutable LevelData.level field is accessible with LevelData.getLevel() getter.
The function class diagram is depicted on Illustration 3.

### 5.1.4 MultiLevelSensor Function

MultiLevelSensor function provides multi-level sensor monitoring. It reports its state when an important event is available. There are no operations. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later. The full function definition is available in the next table.

<table>
<thead>
<tr>
<th>MultiLevelSensor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>Properties</td>
</tr>
<tr>
<td>data</td>
</tr>
<tr>
<td>Types</td>
</tr>
<tr>
<td>light, temperature, flow, pressure, humidity, gas, smoke, door, window, liquid, power, noisiness, rain, other type defined in <code>org.osgi.service.dal.functions.Types</code> or vendor specific type.</td>
</tr>
</tbody>
</table>

MultiLevelSensor and MultiLevelControl are using the same LevelData data structure to provide information about the level. For more details see the definition in MultiLevelControl Function. The function class diagram is depicted on Illustration 3.

### 5.1.5 Meter Function

Meter function can measure metering information.

<table>
<thead>
<tr>
<th>Meter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>Operations</td>
</tr>
<tr>
<td>resetTotal</td>
</tr>
<tr>
<td>Properties</td>
</tr>
<tr>
<td>total</td>
</tr>
<tr>
<td>current</td>
</tr>
<tr>
<td>Service Properties</td>
</tr>
<tr>
<td>dal.meter.flow</td>
</tr>
<tr>
<td>Types</td>
</tr>
<tr>
<td>pressure, gas, power, water, heat, cold, other type defined in <code>org.osgi.service.dal.functions.Types</code> or vendor specific type.</td>
</tr>
</tbody>
</table>

Meter function is using the same LevelData data structure as MultiLevelSensor and MultiLevelControl to provide metering information. For more details see the definition in MultiLevelControl Function. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later. The function class diagram is depicted on Illustration 3.
5.1.6 Alarm Function

Alarm function provides alarm sensor support. There is only one eventable property and no operations. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later.

<table>
<thead>
<tr>
<th>Alarm</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>alarm</td>
<td>Specifies the alarm property name. The property is eventable.</td>
</tr>
</tbody>
</table>

AlarmData data structure is used to provide information about the available alarm. That data object contains the alarm type and severity.

The function class diagram is depicted on Illustration 3.

5.1.7 Keypad Function

Keypad function provides support for keypad control. A keypad typically consists of one or more keys/buttons, which can be discerned. Different types of key presses like short and long press can typically also be detected. There is only one eventable property and no operations. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later.

<table>
<thead>
<tr>
<th>Keypad</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>key</td>
<td>Specifies a property name for a key from the keypad. The property is eventable.</td>
</tr>
</tbody>
</table>

KeypadData data structure is used to provide information when a change with some key from device keypad has occurred. That data object contains the event type, key code and key name. Currently, there are a few predefined event types:

- EVENT_TYPE_PRESSED – used for a key pressed;
- EVENT_TYPE_PRESSED_LONG – used for a long key pressed;
- EVENT_TYPE_PRESSED_DOUBLE – used for a double key pressed;
- EVENT_TYPE_PRESSED_DOUBLE_LONG – used for a double and long key pressed;
- EVENT_TYPE_RELEASED – used for a key released.
- EVENT_TYPE_UNKNOWN – represents an unknown keypad event type.

The function class diagram is depicted on Illustration 3.

5.1.8 WakeUp Function

WakeUp function provides device awake monitoring and management. It's especially applicable to battery-operated devices. Such device can notify the system that it's awake and can receive commands with an event to property PROPERTY_AWAKE. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later.

The device can periodically wake up for commands. The interval can be managed with PROPERTY_WAKE_UP_INTERVAL property.
The application can minimize the power consumption with `sleep()` operation. As a result, the device will sleep and will not receive commands to the next awake.

<table>
<thead>
<tr>
<th>WakeUp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>awake</td>
</tr>
<tr>
<td>wakeUpInterval</td>
</tr>
</tbody>
</table>

### Operations

- **sleep**
  - The device is forced to sleep to minimize the power consumption.

The function class diagram is depicted on Illustration 3.
6 Data Transfer Objects

TODO: Do we need those objects?

7 Javadoc
<table>
<thead>
<tr>
<th>Package Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.servic</td>
<td>14</td>
</tr>
<tr>
<td>e.dal.functions</td>
<td></td>
</tr>
<tr>
<td>Functions 1.0.</td>
<td></td>
</tr>
<tr>
<td>org.osgi.servic</td>
<td>42</td>
</tr>
<tr>
<td>e.dal.functions</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
<tr>
<td>Function Data 1.0.</td>
<td></td>
</tr>
</tbody>
</table>
Package org.osgi.service.dal.functions

Functions 1.0.

See:  
Description

### Interface Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm</td>
<td>Alarm function provides alarm sensor support.</td>
<td>15</td>
</tr>
<tr>
<td>BooleanControl</td>
<td>BooleanControl function provides a boolean control support.</td>
<td>16</td>
</tr>
<tr>
<td>BooleanSensor</td>
<td>BooleanSensor function provides boolean sensor monitoring.</td>
<td>20</td>
</tr>
<tr>
<td>Keypad</td>
<td>Keypad function provides support for keypad control.</td>
<td>22</td>
</tr>
<tr>
<td>Meter</td>
<td>Meter function can measure metering information.</td>
<td>23</td>
</tr>
<tr>
<td>MultiLevelControl</td>
<td>MultiLevelControl function provides multi-level control support.</td>
<td>26</td>
</tr>
<tr>
<td>MultiLevelSensor</td>
<td>MultiLevelSensor function provides multi-level sensor monitoring.</td>
<td>29</td>
</tr>
<tr>
<td>Types</td>
<td>Shares common constants for all functions defined in this package.</td>
<td>31</td>
</tr>
<tr>
<td>WakeUp</td>
<td>WakeUp function provides device awake monitoring and management.</td>
<td>39</td>
</tr>
</tbody>
</table>

Package org.osgi.service.dal.functions Description

Functions 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.dal.functions; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.dal.functions; version="[1.0,1.1)"
```
interface AbstractDevice

Interface Alarm

org.osgi.service.dal.functions

All Superinterfaces:
org.osgi.service.dal.Function

public interface Alarm
extends org.osgi.service.dal.Function

Alarm function provides alarm sensor support. There is only one eventable property and no operations.

See Also:

AlarmData

Field Summary

<table>
<thead>
<tr>
<th>String</th>
<th>PROPERTY_ALARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the alarm property name.</td>
<td></td>
</tr>
</tbody>
</table>

Fields inherited from interface org.osgi.service.dal.Function

SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

Methods inherited from interface org.osgi.service.dal.Function

getOperationMetadata, getPropertyMetadata, getServiceProperty

Field Detail

PROPERTY_ALARM

public static final String PROPERTY_ALARM = "alarm"

Specifies the alarm property name. The property is eventable.

See Also:

AlarmData
Interface BooleanControl

org.osgi.service.dal.functions

All Superinterfaces:
    org.osgi.service.dal.Function

public interface BooleanControl
extends org.osgi.service.dal.Function

BooleanControl function provides a boolean control support. The function state is accessible with `getData()` getter and `setData(boolean)` setter. The state can be reversed with `reverse()` method, can be set to `true` value with `setTrue()` method and can be set to `false` value with `setFalse()` method.

As an example, the function is easily mappable to ZigBee OnOff cluster and Z-Wave Binary Switch command class. The control type can be:

- `Types.TYPE_LIGHT`
- `Types.TYPE_DOOR`
- `Types.TYPE_WINDOW`
- `Types.TYPE_POWER`
- other type defined in `Types`
- custom - vendor specific type

See Also: `BooleanData`

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String OPERATION_REVERSE</td>
<td>17</td>
</tr>
<tr>
<td>Specifies the reverse operation name.</td>
<td></td>
</tr>
<tr>
<td>String OPERATION_SET_FALSE</td>
<td>17</td>
</tr>
<tr>
<td>Specifies the operation name, which sets the control state to <code>false</code> value.</td>
<td></td>
</tr>
<tr>
<td>String OPERATION_SET_TRUE</td>
<td>17</td>
</tr>
<tr>
<td>Specifies the operation name, which sets the control state to <code>true</code> value.</td>
<td></td>
</tr>
<tr>
<td>String PROPERTY_DATA</td>
<td>17</td>
</tr>
<tr>
<td>Specifies the state property name.</td>
<td></td>
</tr>
</tbody>
</table>

Fields inherited from interface org.osgi.service.dal.Function

`SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION`

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BooleanData <code>getData()</code></td>
<td>17</td>
</tr>
<tr>
<td>Returns the current state of BooleanControl.</td>
<td></td>
</tr>
<tr>
<td>void reverse()</td>
<td>18</td>
</tr>
<tr>
<td>Reverses the BooleanControl state.</td>
<td></td>
</tr>
<tr>
<td>void <code>setData(boolean data)</code></td>
<td>18</td>
</tr>
<tr>
<td>Sets the BooleanControl state to the specified value.</td>
<td></td>
</tr>
<tr>
<td>void setFalse()</td>
<td>19</td>
</tr>
<tr>
<td>Sets the BooleanControl state to <code>false</code> value.</td>
<td></td>
</tr>
</tbody>
</table>
void \texttt{setTrue()}\footnote{Sets the \texttt{BooleanControl} state to \texttt{true} value.}

Methods inherited from interface \texttt{org.osgi.service.dal.Function}
- \texttt{getOperationMetadata}, \texttt{getPropertyMetadata}, \texttt{getServiceProperty}

Field Detail

\textbf{OPERATION\_REVERSE}

\texttt{public static final String OPERATION\_REVERSE = "reverse"}

Specifies the reverse operation name. The operation can be executed with \texttt{reverse()} method.

\textbf{OPERATION\_SET\_TRUE}

\texttt{public static final String OPERATION\_SET\_TRUE = "setTrue"}

Specifies the operation name, which sets the control state to \texttt{true} value. The operation can be executed with \texttt{setTrue()} method.

\textbf{OPERATION\_SET\_FALSE}

\texttt{public static final String OPERATION\_SET\_FALSE = "setFalse"}

Specifies the operation name, which sets the control state to \texttt{false} value. The operation can be executed with \texttt{setFalse()} method.

\textbf{PROPERTY\_DATA}

\texttt{public static final String PROPERTY\_DATA = "data"}

Specifies the state property name. The property value is accessible with \texttt{getData()} method.

\texttt{See Also: BooleanData}

Method Detail

data

\texttt{BooleanData getData() throws UnsupportedOperationException, IllegalStateException, org.osgi.service.dal.DeviceException}

Returns the current state of \texttt{BooleanControl}. It's a getter method for \texttt{PROPERTY\_DATA} property.

Returns:
- \texttt{The current state of BooleanControl.}

Throws:
- \texttt{UnsupportedOperationException - If the operation is not supported.}
IllegalStateException - If this function service object has already been unregistered.
org.osgi.service.dal.DeviceException - If an operation error is available.

See Also:
BooleanData, PROPERTY_DATA

setData

void setData(boolean data)
throws UnsupportedOperationException,
IllegalStateException,
org.osgi.service.dal.DeviceException,
IllegalArgumentException

Sets the BooleanControl state to the specified value. It's setter method for PROPERTY_DATA property.

Parameters:
data - The new function value.

Throws:
UnsupportedOperationException - If the operation is not supported.
IllegalStateException - If this function service object has already been unregistered.
org.osgi.service.dal.DeviceException - If an operation error is available.
IllegalArgumentException - If there is an invalid argument.

See Also:
PROPERTY_DATA

reverse

void reverse()
throws UnsupportedOperationException,
IllegalStateException,
or.org.osgi.service.dal.DeviceException

Reverses the BooleanControl state. If the current state represents true value, it'll be reversed to false. If the current state represents false value, it'll be reversed to true. The operation name is OPERATION_REVERSE.

Throws:
UnsupportedOperationException - If the operation is not supported.
IllegalStateException - If this function service object has already been unregistered.
or.org.osgi.service.dal.DeviceException - If an operation error is available.

setTrue

void setTrue()
throws UnsupportedOperationException,
IllegalStateException,
or.org.osgi.service.dal.DeviceException

Sets the BooleanControl state to true value. The operation name is OPERATION_SET_TRUE.

Throws:
UnsupportedOperationException - If the operation is not supported.
IllegalStateException - If this function service object has already been unregistered.
or.org.osgi.service.dal.DeviceException - If an operation error is available.
setFalse

void setFalse()
   throws UnsupportedOperationException,
           IllegalStateException,
           org.osgi.service.dal.DeviceException

Sets the BooleanControl state to false value. The operation name is OPERATION_SET_FALSE.

Throws:
   UnsupportedOperationException - If the operation is not supported.
   IllegalStateException - If this function service object has already been unregistered.
   org.osgi.service.dal.DeviceException - If an operation error is available.
public interface BooleanSensor
extends org.osgi.service.dal.Function

BooleanSensor function provides boolean sensor monitoring. It reports its state when an important event is available. The state is accessible with getData() getter. There are no operations.

As an example, the function is easily mappable to ZigBee Occupancy Sensing cluster and Z-Wave Binary Sensor command class. The sensor type can be:

- Types.TYPE_LIGHT
- Types.TYPE_GAS
- Types.TYPE_SMOKE
- Types.TYPE_DOOR
- Types.TYPE_WINDOW
- Types.TYPE_POWER
- Types.TYPE_RAIN
- Types.TYPE_CONTACT
- Types.TYPE_FIRE
- Types.TYPE_OCCUPANCY
- Types.TYPE_WATER
- Types.TYPE_MOTION
- other type defined in Types
- custom - vendor specific type

See Also:
BooleanData

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String PROPERTY_DATA</td>
<td>21</td>
</tr>
<tr>
<td>Specifies the state property name.</td>
<td></td>
</tr>
</tbody>
</table>

Fields inherited from interface org.osgi.service.dal.Function

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BooleanData getData()</td>
<td>21</td>
</tr>
<tr>
<td>Returns the BooleanSensor current state.</td>
<td></td>
</tr>
</tbody>
</table>

Methods inherited from interface org.osgi.service.dal.Function

getOperationMetadata, getPropertyMetadata, getServiceProperty
**Field Detail**

**PROPERTY_DATA**

public static final String PROPERTY_DATA = "data"

Specifies the state property name. The property value is accessible with `getData()` getter.

**Method Detail**

**getData**

`BooleanData getData()`

throws UnsupportedOperationException, 
UnsupportedOperationException, 
IllegalStateException, 
IllegalStateException, 
org.osgi.service.dal.DeviceException

Returns the `BooleanSensor` current state. It's a getter method for `PROPERTY_DATA` property.

- **Returns:** The `BooleanSensor` current state.
- **Throws:** 
  - `UnsupportedOperationException` - If the operation is not supported.
  - `IllegalStateException` - If this function service object has already been unregistered.
  - `org.osgi.service.dal.DeviceException` - If an operation error is available.
- **See Also:** `BooleanData`
**Interface Keypad**

org.osgi.service.dal.functions

All Superinterfaces:
org.osgi.service.dal.Function

```
public interface Keypad
extends org.osgi.service.dal.Function
```

Keypad function provides support for keypad control. A keypad typically consists of one or more keys/buttons, which can be discerned. Different types of key presses like short and long press can typically also be detected. There is only one eventable property and no operations.

Keypad can enumerate all supported keys in the `key` property metadata, org.osgi.service.dal.PropertyMetadata.getEnumValues(String). KeypadData event type will be `KeypadData.EVENT_TYPE_UNKNOWN` in this case.

See Also:
KeypadData

### Field Summary

<table>
<thead>
<tr>
<th>String</th>
<th>PROPERTY_KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specifies a property name for a key from the keypad.</td>
</tr>
</tbody>
</table>

Page 22

### Fields inherited from interface org.osgi.service.dal.Function

SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

### Methods inherited from interface org.osgi.service.dal.Function

getOperationMetadata, getPropertyMetadata, getServiceProperty

### Field Detail

**PROPERTY_KEY**

```
public static final String PROPERTY_KEY = "key"
```

Specifies a property name for a key from the keypad. The property is eventable.

See Also:
KeypadData
public interface Meter
extends org.osgi.service.dal.Function

Meter function can measure metering information. The function provides three properties and one operation:

- PROPERTY_CURRENT
- property accessible with getCurrent() getter;
- PROPERTY_TOTAL
- property accessible with getTotal() getter;
- SERVICE_FLOW
- property accessible with getTotal() getter;
- OPERATION_RESET_TOTAL
- operation can be executed with resetTotal().

As an example, the function is easily mappable to ZigBee Simple Metering cluster and Z-Wave Meter command class. The sensor type can be:

- Types.TYPE_PRESSURE
- Types.TYPE_GAS
- Types.TYPE_POWER
- Types.TYPE_WATER
- Types.TYPE_HEAT
- Types.TYPE_COLD
- other type defined in Types
- custom - vendor specific type

See Also:
LevelData

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String FLOW_IN</td>
<td>24</td>
</tr>
<tr>
<td>Represents the metering consumption flow.</td>
<td></td>
</tr>
<tr>
<td>String FLOW_OUT</td>
<td>24</td>
</tr>
<tr>
<td>Represents the metering generation flow.</td>
<td></td>
</tr>
<tr>
<td>String OPERATION_RESET_TOTAL</td>
<td>25</td>
</tr>
<tr>
<td>Specifies the reset total operation name.</td>
<td></td>
</tr>
<tr>
<td>String PROPERTY_CURRENT</td>
<td>24</td>
</tr>
<tr>
<td>Specifies the current consumption property name.</td>
<td></td>
</tr>
<tr>
<td>String PROPERTY_TOTAL</td>
<td>24</td>
</tr>
<tr>
<td>Specifies the total consumption property name.</td>
<td></td>
</tr>
<tr>
<td>String SERVICE_FLOW</td>
<td>24</td>
</tr>
<tr>
<td>The service property value contains the metering flow.</td>
<td></td>
</tr>
</tbody>
</table>

Fields inherited from interface org.osgi.service.dal.Function

SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES,
SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION
Interface DeviceFunction

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LevelData getCurrent()</td>
<td>25</td>
</tr>
<tr>
<td>Returns the current metering info.</td>
<td></td>
</tr>
<tr>
<td>LevelData getTotal()</td>
<td>25</td>
</tr>
<tr>
<td>Returns the total metering info.</td>
<td></td>
</tr>
<tr>
<td>void resetTotal()</td>
<td>25</td>
</tr>
<tr>
<td>Resets the total metering info.</td>
<td></td>
</tr>
</tbody>
</table>

Methods inherited from interface org.osgi.service.dal.Function
getOperationMetadata, getPropertyMetadata, getServiceProperty

Field Detail

FLOW_IN

public static final String FLOW_IN = "in"

Represents the metering consumption flow. It can be used as SERVICE_FLOW property value.

FLOW_OUT

public static final String FLOW_OUT = "out"

Represents the metering generation flow. It can be used as SERVICE_FLOW property value.

SERVICE_FLOW

public static final String SERVICE_FLOW = "dal.meter.flow"

The service property value contains the metering flow. It's an optional property and available only if it's supported by the meter. The value type is java.lang.String. Possible property values:

- FLOW_IN
- FLOW_OUT

PROPERTY_CURRENT

public static final String PROPERTY_CURRENT = "current"

Specifies the current consumption property name. The property can be read with getCurrent() getter.

PROPERTY_TOTAL

public static final String PROPERTY_TOTAL = "total"

Specifies the total consumption property name. It has been measured since the last call of resetTotal() or device initial run. The property can be read with getTotal() getter.
OPERATION_RESET_TOTAL

```java
public static final String OPERATION_RESET_TOTAL = "resetTotal"
```

Specifies the reset total operation name. The operation can be executed with `resetTotal()` method.

**Method Detail**

### getCurrent

```java
LevelData getCurrent()
```

Throws `UnsupportedOperationException`, `IllegalStateException`, `org.osgi.service.dal.DeviceException`

Returns the current metering info. It's a getter method for `PROPERTY_CURRENT` property.

**Returns:**
The current metering info.

**Throws:**
- `UnsupportedOperationException` - If the operation is not supported.
- `IllegalStateException` - If this function service object has already been unregistered.
- `org.osgi.service.dal.DeviceException` - If an operation error is available.

**See Also:**
`LevelData`

### getTotal

```java
LevelData getTotal()
```

Throws `UnsupportedOperationException`, `IllegalStateException`, `org.osgi.service.dal.DeviceException`

Returns the total metering info. It's a getter method for `PROPERTY_TOTAL` property.

**Returns:**
The total metering info.

**Throws:**
- `UnsupportedOperationException` - If the operation is not supported.
- `IllegalStateException` - If this function service object has already been unregistered.
- `org.osgi.service.dal.DeviceException` - If an operation error is available.

**See Also:**
`LevelData`

### resetTotal

```java
void resetTotal()
```

Throws `UnsupportedOperationException`, `IllegalStateException`, `org.osgi.service.dal.DeviceException`

Resets the total metering info.

**Throws:**
- `UnsupportedOperationException` - If the operation is not supported.
- `IllegalStateException` - If this function service object has already been unregistered.
- `org.osgi.service.dal.DeviceException` - If an operation error is available.
public interface MultiLevelControl
extends org.osgi.service.dal.Function

MultiLevelControl function provides multi-level control support. The function level is accessible with `getData()` getter, `setData(BigDecimal)` setter and `setData(BigDecimal, String)` setter.

As an example, the function is easily mappable to ZigBee Level Control and Z-Wave Multilevel Switch command class. The control type can be:

- Types.TYPE_LIGHT
- Types.TYPE_TEMPERATURE
- Types.TYPE_FLOW
- Types.TYPE_PRESSURE
- Types.TYPE_HUMIDITY
- Types.TYPE_GAS
- Types.TYPE_SMOKE
- Types.TYPE_DOOR
- Types.TYPE_WINDOW
- Types.TYPE_LIQUID
- Types.TYPE_POWER
- Types.TYPE_NOISINESS
- other type defined in Types
- custom - vendor specific type

See Also: LevelData

Field Summary

<table>
<thead>
<tr>
<th>String</th>
<th>PROPERTY_DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specifies the level property name.</td>
</tr>
</tbody>
</table>

Fields inherited from interface org.osgi.service.dal.Function
SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

Method Summary

<table>
<thead>
<tr>
<th>LevelData</th>
<th><code>getData()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns MultiLevelControl level.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th><code>setData(BigDecimal level)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets MultiLevelControl level to the specified value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th><code>setData(BigDecimal level, String unit)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets MultiLevelControl level according to the specified unit.</td>
</tr>
</tbody>
</table>

Methods inherited from interface org.osgi.service.dal.Function
getOperationMetadata, getPropertyMetadata, getServiceProperty
**Interface FunctionalDevice**

### Field Detail

**PROPERTY_DATA**

public static final String PROPERTY_DATA = "data"

Specifies the level property name. The property can be read with `getData()` getter and can be set with `setData(BigDecimal)` or `setData(BigDecimal, String)` setters.

### Method Detail

**getData**

```java
LevelData getData() throws UnsupportedOperationException, IllegalArgumentException, IllegalStateException, org.osgi.service.dal.DeviceException
```

Returns MultiLevelControl level. It's a getter method for `PROPERTY_DATA` property.

**Returns:**
MultiLevelControl level.

**Throws:**
- UnsupportedOperationException - If the operation is not supported.
- IllegalStateException - If this function service object has already been unregistered.
- org.osgi.service.dal.DeviceException - If an operation error is available.

**See Also:**
LevelData

**setData**

```java
void setData(BigDecimal level) throws UnsupportedOperationException, IllegalArgumentException, IllegalStateException, org.osgi.service.dal.DeviceException, IllegalArgumentException
```

Sets MultiLevelControl level to the specified value. It's a setter method for `PROPERTY_DATA` property.

**Parameters:**
level - The new control level.

**Throws:**
- UnsupportedOperationException - If the operation is not supported.
- IllegalStateException - If this function service object has already been unregistered.
- org.osgi.service.dal.DeviceException - If an operation error is available.
- IllegalArgumentException - If there is an invalid argument.
Interface FunctionalDevice

setData

void setData(BigDecimal level,
       String unit)
    throws UnsupportedOperationException,
           IllegalStateException,
           org.osgi.service.dal.DeviceException,
           IllegalArgumentException

Sets MultiLevelControl level according to the specified unit. It's a setter method for PROPERTY_DATA property.

Parameters:
  level - The new control level.
  unit - The level unit.

Throws:
  UnsupportedOperationException - If the operation is not supported.
  IllegalStateException - If this function service object has already been unregistered.
  org.osgi.service.dal.DeviceException - If an operation error is available.
  IllegalArgumentException - If there is an invalid argument.
## Interface MultiLevelSensor

### org.osgi.service.dal.functions

All Superinterfaces:

- org.osgi.service.dal.Function

```java
public interface MultiLevelSensor extends org.osgi.service.dal.Function
```

MultiLevelSensor function provides multi-level sensor monitoring. It reports its state when an important event is available. The state is accessible with `getData()` getter. There are no operations.

As an example, the function is easily mappable to ZigBee Illuminance Measurement, Temperature Measurement, Pressure Measurement, Flow Measurement and Relative Humidity Measurement cluster and Z-Wave Multilevel Sensor command class. The sensor type can be:

- `Types.TYPE_LIGHT`
- `Types.TYPE_TEMPERATURE`
- `Types.TYPE_FLOW`
- `Types.TYPE_PRESSURE`
- `Types.TYPE_HUMIDITY`
- `Types.TYPE_GAS`
- `Types.TYPE_SMOKE`
- `Types.TYPE_DOOR`
- `Types.TYPE_WINDOW`
- `Types.TYPE_LIQUID`
- `Types.TYPE_POWER`
- `Types.TYPE_NOISINESS`
- `Types.TYPE_RAIN`
- other type defined in `Types`
- custom - vendor specific type

See Also:

- `LevelData`

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROPERTY_DATA</td>
<td>String</td>
<td>Specifies the state property name.</td>
</tr>
</tbody>
</table>

### Fields inherited from interface org.osgi.service.dal.Function

- `SERVICE_DESCRIPTION`, `SERVICE_DEVICE_UID`, `SERVICE_OPERATION_NAMES`, `SERVICE_PROPERTY_NAMES`, `SERVICE_REFERENCE_UIDS`, `SERVICE_TYPE`, `SERVICE_UID`, `SERVICE_VERSION`

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LevelData.getData()</td>
<td>Returns the MultiLevelSensor current state.</td>
</tr>
</tbody>
</table>

### Methods inherited from interface org.osgi.service.dal.Function

- `getOperationMetadata`, `getPropertyMetadata`, `getServiceProperty`
Field Detail

PROPERTY_DATA

public static final String PROPERTY_DATA = "data"

Specifies the state property name. The property can be read with `getData()` getter.

See Also:
LevelData

Method Detail

getData

```java
LevelData getData() throws UnsupportedOperationException, IllegalStateException, org.osgi.service.dal.DeviceException
```

Returns the MultiLevelSensor current state. It's a getter method for PROPERTY_DATA property.

Returns:
The MultiLevelSensor current state.

Throws:
UnsupportedOperationException - If the operation is not supported.
IllegalStateException - If this function service object has already been unregistered.
org.osgi.service.dal.DeviceException - If an operation error is available.

See Also:
LevelData
public interface Types

Shares common constants for all functions defined in this package. The defined function types are mapped as follow:

- **TYPE_LIGHT** - MultiLevelControl, MultiLevelSensor, BooleanSensor and BooleanControl
- **TYPE_TEMPERATURE** - MultiLevelControl and MultiLevelSensor
- **TYPE_FLOW** - MultiLevelControl and MultiLevelSensor
- **TYPE_PRESSURE** - MultiLevelControl, MultiLevelSensor and Meter
- **TYPE_HUMIDITY** - MultiLevelControl and MultiLevelSensor
- **TYPE_GAS** - MultiLevelControl, MultiLevelSensor, BooleanSensor and Meter
- **TYPE_SMOKE** - MultiLevelControl, MultiLevelSensor, BooleanSensor and BooleanControl
- **TYPE_WINDOW** - MultiLevelControl, MultiLevelSensor, BooleanSensor and BooleanControl
- **TYPE_LIQUID** - MultiLevelControl and MultiLevelSensor
- **TYPE_POWER** - MultiLevelControl, MultiLevelSensor, BooleanSensor, BooleanControl and Meter
- **TYPE_NOISINESS** - MultiLevelControl and MultiLevelSensor
- **TYPE_RAIN** - MultiLevelSensor and BinarySensor
- **TYPE_CONTACT** - BinarySensor
- **TYPE_FIRE** - BinarySensor
- **TYPE_OCCUPANCY** - BinarySensor
- **TYPE_WATER** - BinarySensor and Meter
- **TYPE_MOTION** - BinarySensor
- **TYPE_HEAT** - Meter
- **TYPE_COLD** - Meter

The mapping is not mandatory. The function can use custom defined types.

### Field Summary

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td><strong>TYPE_COLD</strong></td>
<td>Meter - indicates that the Meter measures thermal energy provided by a source. This type can be specified as a value of org.osgi.service.dal.Function.SERVICE_TYPE.</td>
</tr>
<tr>
<td>String</td>
<td><strong>TYPE_CONTACT</strong></td>
<td>BinarySensor - indicates that the BinarySensor can detect contact.</td>
</tr>
<tr>
<td>String</td>
<td><strong>TYPE_DOOR</strong></td>
<td>MultiLevelControl - indicates that the MultiLevelControl can control the door position.</td>
</tr>
</tbody>
</table>
### Interface BinarySwitch

<table>
<thead>
<tr>
<th>String</th>
<th>The function type is applicable to:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TYPE_FIRE</strong></td>
<td>- BinarySensor - indicates that the BinarySensor can detect fire.</td>
</tr>
<tr>
<td><strong>TYPE_FLOW</strong></td>
<td>- MultiLevelControl - indicates that the MultiLevelControl can control the flow level.</td>
</tr>
<tr>
<td><strong>TYPE_GAS</strong></td>
<td>- MultiLevelControl - indicates that the MultiLevelControl can control the gas level.</td>
</tr>
<tr>
<td><strong>TYPE_HEAT</strong></td>
<td>- Meter - indicates that the Meter measures thermal energy provided by a source.</td>
</tr>
<tr>
<td><strong>TYPE_HUMIDITY</strong></td>
<td>This type can be specified as a value of org.osgi.service.dal.Function.SERVICE_TYPE.</td>
</tr>
<tr>
<td><strong>TYPE_LIGHT</strong></td>
<td>- MultiLevelControl - indicates that the MultiLevelControl can control the humidity level.</td>
</tr>
<tr>
<td><strong>TYPE_LIQUID</strong></td>
<td>- MultiLevelControl - indicates that the MultiLevelControl can control the liquid level.</td>
</tr>
<tr>
<td><strong>TYPE_MOTION</strong></td>
<td>- BinarySensor - indicates that the BinarySensor can detect motion.</td>
</tr>
<tr>
<td><strong>TYPE_NOISINESS</strong></td>
<td>- MultiLevelControl - indicates that the MultiLevelControl can control the noise level.</td>
</tr>
<tr>
<td>Interface BinarySwitch</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>String TYPE_OCCUPANCY</td>
<td></td>
</tr>
<tr>
<td>The function type is applicable to:</td>
<td></td>
</tr>
<tr>
<td>- BinarySensor - indicates that the BinarySensor can detect presence.</td>
<td></td>
</tr>
<tr>
<td>String TYPE_POWER</td>
<td></td>
</tr>
<tr>
<td>The function type is applicable to:</td>
<td></td>
</tr>
<tr>
<td>- MultiLevelControl - indicates that the MultiLevelControl can control the power level.</td>
<td></td>
</tr>
<tr>
<td>String TYPE_PRESSURE</td>
<td></td>
</tr>
<tr>
<td>The function type is applicable to:</td>
<td></td>
</tr>
<tr>
<td>- MultiLevelControl - indicates that the MultiLevelControl can control the pressure level.</td>
<td></td>
</tr>
<tr>
<td>String TYPE_RAIN</td>
<td></td>
</tr>
<tr>
<td>The function type is applicable to:</td>
<td></td>
</tr>
<tr>
<td>- MultiLevelSensor - indicates that the MultiLevelSensor can monitor the rain rate.</td>
<td></td>
</tr>
<tr>
<td>String TYPE_SMOKE</td>
<td></td>
</tr>
<tr>
<td>The function type is applicable to:</td>
<td></td>
</tr>
<tr>
<td>- MultiLevelControl - indicates that the MultiLevelControl can control the smoke level.</td>
<td></td>
</tr>
<tr>
<td>String TYPE_TEMPERATURE</td>
<td></td>
</tr>
<tr>
<td>The function type is applicable to:</td>
<td></td>
</tr>
<tr>
<td>- MultiLevelControl - indicates that the MultiLevelControl can control temperature devices.</td>
<td></td>
</tr>
<tr>
<td>String TYPE_WATER</td>
<td></td>
</tr>
<tr>
<td>The function type is applicable to:</td>
<td></td>
</tr>
<tr>
<td>- BinarySensor - indicates that the BinarySensor can detect water leak.</td>
<td></td>
</tr>
<tr>
<td>String TYPE_WINDOW</td>
<td></td>
</tr>
<tr>
<td>The function type is applicable to:</td>
<td></td>
</tr>
<tr>
<td>- MultiLevelControl - indicates that the MultiLevelControl can control the window position.</td>
<td></td>
</tr>
</tbody>
</table>

### Field Detail

**TYPE_LIGHT**

```java
public static final String TYPE_LIGHT = "light"
```

The function type is applicable to:
- MultiLevelControl - indicates that the MultiLevelControl can control light devices. Usually, such devices are called dimmable. MultiLevelControl minimum value can switch off the device and MultiLevelControl maximum value can increase the device light to the maximum possible value.
- MultiLevelSensor - indicates that the sensor can monitor the light level.
- BinarySensor - indicates that the BinarySensor can detect light. true state means that there is light. false state means that there is no light.
- BinaryControl - indicates that there is a light device control. true state means that the light device will be turned on. false state means that the light device will be turned off.

This type can be specified as a value of org.osgi.service.dal.Function.SERVICE_TYPE.

**TYPE_TEMPERATURE**

```java
public static final String TYPE_TEMPERATURE = "temperature"
```

The function type is applicable to:

- MultiLevelControl - indicates that the MultiLevelControl can control temperature devices. For example, such device can be thermostat. MultiLevelControl minimum value is the lowest supported temperature. MultiLevelControl maximum value is the highest supported temperature.
- MultiLevelSensor - indicates that the sensor can monitor the temperature.

This type can be specified as a value of org.osgi.service.dal.Function.SERVICE_TYPE.

**TYPE_FLOW**

```java
public static final String TYPE_FLOW = "flow"
```

The function type is applicable to:

- MultiLevelControl - indicates that the MultiLevelControl can control the flow level. MultiLevelControl minimum value is the minimum supported flow level. MultiLevelControl maximum value is the maximum supported flow level.
- MultiLevelSensor - indicates that the sensor can monitor the flow level.

This type can be specified as a value of org.osgi.service.dal.Function.SERVICE_TYPE.

**TYPE_PRESSURE**

```java
public static final String TYPE_PRESSURE = "pressure"
```

The function type is applicable to:

- MultiLevelControl - indicates that the MultiLevelControl can control the pressure level. MultiLevelControl minimum value is the lowest supported pressure level. MultiLevelControl maximum value is the highest supported pressure level.
- MultiLevelSensor - indicates that the sensor can monitor the pressure level.
- Meter - Indicates that the Meter measures pressure.

This type can be specified as a value of org.osgi.service.dal.Function.SERVICE_TYPE.

**TYPE_HUMIDITY**

```java
public static final String TYPE_HUMIDITY = "humidity"
```
The function type is applicable to:

- **MultiLevelControl** - indicates that the MultiLevelControl can control the humidity level. It's typical functionality for HVAC (heating, ventilation, and air conditioning) devices. MultiLevelControl minimum value is the lowest supported humidity level. MultiLevelControl maximum value is the highest supported humidity level.
- **MultiLevelSensor** - indicates that the sensor can monitor the humidity level.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

---

**TYPE_GAS**

```java
public static final String TYPE_GAS = "gas"
```

The function type is applicable to:

- **MultiLevelControl** - indicates that the MultiLevelControl can control the gas level. MultiLevelControl minimum value is the lowest supported gas level. MultiLevelControl maximum value is the highest supported gas level.
- **MultiLevelSensor** - indicates that the sensor can monitor the gas level.
- **BinarySensor** - indicates that the BinarySensor supports gas detection. true state means there is gas. false state means that there is no gas.
- **Meter** - indicates that the Meter measures the gas consumption.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

---

**TYPE_SMOKE**

```java
public static final String TYPE_SMOKE = "smoke"
```

The function type is applicable to:

- **MultiLevelControl** - indicates that the MultiLevelControl can control the smoke level. MultiLevelControl minimum value is the lowest supported smoke level. MultiLevelControl maximum value is the highest supported smoke level.
- **MultiLevelSensor** - indicates that the sensor can monitor the smoke level.
- **BinarySensor** - indicates that the BinarySensor can detect smoke. true state means that there is smoke. false state means that there is no rain.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

---

**TYPE_DOOR**

```java
public static final String TYPE_DOOR = "door"
```

The function type is applicable to:

- **MultiLevelControl** - indicates that the MultiLevelControl can control the door position. MultiLevelControl minimum value can completely close the door. MultiLevelControl maximum value can open the door to the maximum allowed position.
- **MultiLevelSensor** - indicates that the sensor can monitor the door position.
- **BinarySensor** - indicates that the BinarySensor can detect the door state. true state means that the door is opened. false state means that the door is closed.
- **BinaryControl** - indicates that there is a door position control. true state means that the door will be opened. false state means that the door will be closed.
Interface BinarySwitch

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

### TYPE_WINDOW

public static final String `TYPE_WINDOW` = "window"

The function type is applicable to:

- **MultiLevelControl** - indicates that the `MultiLevelControl` can control the window position. `MultiLevelControl` minimum value can completely close the window. `MultiLevelControl` maximum value can open the window to the maximum allowed position.
- **MultiLevelSensor** - indicates that the sensor can monitor the window position.
- **BinarySensor** - indicates that the `BinarySensor` can window state. `true` state means that the window is opened. `false` state means that the window is closed.
- **BinaryControl** - indicates that there is a window position control. `true` state means that the window will be opened. `false` state means that the the window will be closed.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

### TYPE_LIQUID

public static final String `TYPE_LIQUID` = "liquid"

The function type is applicable to:

- **MultiLevelControl** - indicates that the `MultiLevelControl` can control the liquid level. `MultiLevelControl` minimum value is the lowest supported liquid level. `MultiLevelControl` maximum value is the highest supported liquid level.
- **MultiLevelSensor** - indicates that the sensor can monitor the liquid level.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

### TYPE_POWER

public static final String `TYPE_POWER` = "power"

The function type is applicable to:

- **MultiLevelControl** - indicates that the `MultiLevelControl` can control the power level. `MultiLevelControl` minimum value is the lowest supported power level. `MultiLevelControl` maximum value is the highest supported power level.
- **MultiLevelSensor** - indicates that the sensor can monitor the power level.
- **BinarySensor** - indicates that the `BinarySensor` can detect motion. `true` state means that there is power restore. `false` state means that there is power cut.
- **BinaryControl** - indicates that there is electricity control. `true` state means that the power will be restored. `false` state means that the power will be cut.
- **Meter** - indicates that the `Meter` measures the power consumption.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

### TYPE_NOISINESS

public static final String `TYPE_NOISINESS` = "noisiness"
Interface BinarySwitch

The function type is applicable to:

- **MultiLevelControl** - indicates that the MultiLevelControl can control the noise level. MultiLevelControl minimum value is the lowest supported noise level. MultiLevelControl maximum value is the highest supported noise level.
- **MultiLevelSensor** - indicates that the sensor can monitor the noise level.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

### TYPE_RAIN

```java
public static final String TYPE_RAIN = "rain"
```

The function type is applicable to:

- **MultiLevelSensor** - indicates that the MultiLevelSensor can monitor the rain rate. It's not applicable to MultiLevelControl.
- **BinarySensor** - indicates that the BinarySensor can detect rain. true state means that there is rain. false state means that there is no rain.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

### TYPE_CONTACT

```java
public static final String TYPE_CONTACT = "contact"
```

The function type is applicable to:

- **BinarySensor** - indicates that the BinarySensor can detect contact. true state means that there is contact. false state means that there is no contact.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

### TYPE_FIRE

```java
public static final String TYPE_FIRE = "fire"
```

The function type is applicable to:

- **BinarySensor** - indicates that the BinarySensor can detect fire. true state means that there is fire. false state means that there is no fire.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

### TYPE_OCCUPANCY

```java
public static final String TYPE_OCCUPANCY = "occupancy"
```

The function type is applicable to:

- **BinarySensor** - indicates that the BinarySensor can detect presence. true state means that someone is detected. false state means that nobody is detected.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`. 
### TYPE_WATER

```java
public static final String TYPE_WATER = "water"
```

The function type is applicable to:

- **BinarySensor** - indicates that the `BinarySensor` can detect water leak. `true` state means that there is water leak, `false` state means that there is no water leak.
- **Meter** - indicates that the `Meter` measures water consumption.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

### TYPE_MOTION

```java
public static final String TYPE_MOTION = "motion"
```

The function type is applicable to:

- **BinarySensor** - indicates that the `BinarySensor` can detect motion. `true` state means that there is motion detection, `false` state means that there is no motion detection.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

### TYPE_HEAT

```java
public static final String TYPE_HEAT = "heat"
```

The function type is applicable to:

- **Meter** - indicates that the `Meter` measures thermal energy provided by a source.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

### TYPE_COLD

```java
public static final String TYPE_COLD = "cold"
```

The function type is applicable to:

- **Meter** - indicates that the `Meter` measures thermal energy provided by a source.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`. 
## Interface WakeUp

**org.osgi.service.dal.functions**

### All Superinterfaces:

org.osgi.service.dal.Function

```java
public interface WakeUp extends org.osgi.service.dal.Function
```

*WakeUp* function provides device awake monitoring and management. It's especially applicable to battery-operated devices. Such device can notify the system that it's awake and can receive commands with an event to property `PROPERTY_AWAKE`.

The device can periodically wake up for commands. The interval can be managed with `PROPERTY_WAKE_UP_INTERVAL` property.

The application can minimize the power consumption with `sleep()` operation. As a result, the device will sleep and will not receive commands to the next awake.

**See Also:**

[LevelData](#), [BooleanData](#)

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String <code>OPERATION_SLEEP</code></td>
<td>40</td>
</tr>
<tr>
<td>Specifies the sleep operation name.</td>
<td></td>
</tr>
<tr>
<td>String <code>PROPERTY_AWAKE</code></td>
<td>40</td>
</tr>
<tr>
<td>Specifies the awake property name.</td>
<td></td>
</tr>
<tr>
<td>String <code>PROPERTY_WAKE_UP_INTERVAL</code></td>
<td>40</td>
</tr>
<tr>
<td>Specifies the wake up interval.</td>
<td></td>
</tr>
</tbody>
</table>

### Fields inherited from interface org.osgi.service.dal.Function

SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LevelData getWakeUpInterval()</code></td>
<td>40</td>
</tr>
<tr>
<td>Returns the current wake up interval.</td>
<td></td>
</tr>
<tr>
<td><code>void setWakeUpInterval(BigDecimal interval)</code></td>
<td>41</td>
</tr>
<tr>
<td>Sets wake up interval according to the default unit.</td>
<td></td>
</tr>
<tr>
<td><code>void setWakeUpInterval(BigDecimal interval, String unit)</code></td>
<td>41</td>
</tr>
<tr>
<td>Sets wake up interval according to the specified unit.</td>
<td></td>
</tr>
<tr>
<td><code>void sleep()</code></td>
<td>41</td>
</tr>
<tr>
<td>The device is forced to sleep to minimize the power consumption.</td>
<td></td>
</tr>
</tbody>
</table>

### Methods inherited from interface org.osgi.service.dal.Function

getOperationMetadata, getPropertyMetadata, getServiceProperty
### Field Detail

**PROPERTY_AWAKE**

```java
public static final String PROPERTY_AWAKE = "awake"
```

Specifies the awake property name. The property access type can be `org.osgi.service.dal.PropertyMetadata.PROPERTY_ACCESS_EVENTABLE`. If the device is awake, it will trigger a property event.

The property value type is `BooleanData`. The boolean data is always `true`. It marks that the device is awake.

**PROPERTY_WAKE_UP_INTERVAL**

```java
public static final String PROPERTY_WAKE_UP_INTERVAL = "wakeUpInterval"
```

Specifies the wake up interval. The device can periodically wake up and receive commands. That interval is managed by this property. The current property value is available with `getWakeUpInterval()` and can be modified with `setWakeUpInterval(BigDecimal)` and `setWakeUpInterval(BigDecimal, String)`.

**OPERATION_SLEEP**

```java
public static final String OPERATION_SLEEP = "sleep"
```

Specifies the sleep operation name. The operation can be executed with `sleep()` method.

### Method Detail

**getWakeUpInterval**

```java
LevelData getWakeUpInterval()
```  

Throws:

- `UnsupportedOperationException` - If the operation is not supported.
- `IllegalStateException` - If this function service object has already been unregistered.
- `org.osgi.service.dal.DeviceException` - If an operation error is available.

Returns the current wake up interval. It's a getter method for `PROPERTY_WAKE_UP_INTERVAL` property. The device can periodically wake up and receive command based on this interval.

The interval can be measured in different units like hours, minutes, seconds etc. The unit is specified in `LevelData` instance.

**See Also:**

- `LevelData`
setWakeUpInterval

```java
void setWakeUpInterval(BigDecimal interval)
    throws UnsupportedOperationException,
            IllegalStateException,
            org.osgi.service.dal.DeviceException,
            IllegalArgumentException
```

Sets wake up interval according to the default unit. It's a setter method for `PROPERTY_WAKE_UP_INTERVAL` property. The device can periodically wake up and receive command based on this interval.

**Parameters:**
- `interval` - The new wake up interval.

**Throws:**
- `UnsupportedOperationException` - If the operation is not supported.
- `IllegalStateException` - If this function service object has already been unregistered.
- `org.osgi.service.dal.DeviceException` - If an operation error is available.
- `IllegalArgumentException` - If there is an invalid argument.

---

setWakeUpInterval

```java
void setWakeUpInterval(BigDecimal interval, String unit)
    throws UnsupportedOperationException,
            IllegalStateException,
            org.osgi.service.dal.DeviceException,
            IllegalArgumentException
```

Sets wake up interval according to the specified unit. It's a setter method for `PROPERTY_WAKE_UP_INTERVAL` property. The device can periodically wake up and receive command based on this interval.

**Parameters:**
- `interval` - The new wake up interval.
- `unit` - The interval unit.

**Throws:**
- `UnsupportedOperationException` - If the operation is not supported.
- `IllegalStateException` - If this function service object has already been unregistered.
- `org.osgi.service.dal.DeviceException` - If an operation error is available.
- `IllegalArgumentException` - If there is an invalid argument.

---

sleep

```java
void sleep()
    throws UnsupportedOperationException,
            IllegalStateException,
            org.osgi.service.dal.DeviceException
```

The device is forced to sleep to minimize the power consumption.

**Throws:**
- `UnsupportedOperationException` - If the operation is not supported.
- `IllegalStateException` - If this function service object has already been unregistered.
- `org.osgi.service.dal.DeviceException` - If an operation error is available.
Package org.osgi.service.dal.functions.data

Function Data 1.0.

See:  
Description

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlarmData</td>
<td>43</td>
</tr>
<tr>
<td>BooleanData</td>
<td>50</td>
</tr>
<tr>
<td>KeypadData</td>
<td>54</td>
</tr>
<tr>
<td>LevelData</td>
<td>60</td>
</tr>
</tbody>
</table>

AlarmData Function alarm data.

BooleanData Function boolean data wrapper.

KeypadData Represents a keypad event data that is collected when a change with some key from device keypad has occurred.

LevelData Function level data wrapper.

Package org.osgi.service.dal.functions.data Description

Function Data 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.dal.functions.data; version="[1.0,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.service.dal.functions.data; version="[1.0,1.1)"
Class AlarmData

org.osgi.service.dal.functions.data

doesn't support unit. The alarm type is mapped to FunctionData value.

See Also:
   Alarm, org.osgi.service.dal.FunctionData

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static String</td>
<td>FIELD_SEVERITY</td>
<td>Represents the severity field name.</td>
</tr>
<tr>
<td>static String</td>
<td>FIELD_TYPE</td>
<td>Represents the type field name.</td>
</tr>
<tr>
<td>int severity</td>
<td></td>
<td>Represents the alarm severity.</td>
</tr>
<tr>
<td>static int</td>
<td>SEVERITY_HIGH</td>
<td>The severity rating indicates that there is an alarm with high priority.</td>
</tr>
<tr>
<td>static int</td>
<td>SEVERITY_LOW</td>
<td>The severity rating indicates that there is an alarm with lowest priority.</td>
</tr>
<tr>
<td>static int</td>
<td>SEVERITY_MEDIUM</td>
<td>The severity rating indicates that there is an alarm with medium priority.</td>
</tr>
<tr>
<td>static int</td>
<td>SEVERITY_NONE</td>
<td>The severity constant indicates that there is no severity rating for this alarm.</td>
</tr>
<tr>
<td>static int</td>
<td>SEVERITY_URGENT</td>
<td>The severity rating indicates that there an urgent alarm.</td>
</tr>
<tr>
<td>static int</td>
<td>TYPE_COLD</td>
<td>The alarm type indicates that temperature is too low.</td>
</tr>
<tr>
<td>static int</td>
<td>TYPE_GAS_CO</td>
<td>The alarm type indicates that carbon monoxide is detected.</td>
</tr>
<tr>
<td>static int</td>
<td>TYPE_GAS_CO2</td>
<td>The alarm type indicates that carbon dioxide is detected.</td>
</tr>
<tr>
<td>static int</td>
<td>TYPE_HEAT</td>
<td>The alarm type indicates that temperature is too high.</td>
</tr>
<tr>
<td>static int</td>
<td>TYPE_HW_FAIL</td>
<td>The alarm type indicates that there is hardware failure.</td>
</tr>
<tr>
<td>static int</td>
<td>TYPE_POWER_FAIL</td>
<td>The alarm type indicates a power cut.</td>
</tr>
</tbody>
</table>

public class AlarmData
extends org.osgi.service.dal.FunctionData

Function alarm data. It cares about the alarm type, severity, timestamp and additional metadata.
static int TYPE_SMOKE
    The alarm type indicates that smoke is detected. 45

static int TYPE_SW_FAIL
    The alarm type indicates that there is software failure. 46

static int TYPE_WATER
    The alarm type indicates that water leak is detected. 45

Fields inherited from class org.osgi.service.dal.FunctionData
FIELD_METADATA, FIELD_TIMESTAMP, META_INFO_DESCRIPTION, metadata, timestamp

Constructor Summary

AlarmData (Map fields)
    Constructs new AlarmData instance with the specified field values. 47

AlarmData (long timestamp, Map metadata, int severity, int type)
    Constructs new AlarmData instance with the specified arguments. 47

Method Summary

int compareTo (Object o)
    Compares this AlarmData instance with the given argument. 48

boolean equals (Object other)
    Two AlarmData instances are equal if they contain equal metadata, timestamp, type and severity. 48

int getSeverity ()
    Returns the alarm severity. 48

int getType ()
    Returns the alarm type. 47

int hashCode ()
    Returns the hash code for this AlarmData object. 48

Methods inherited from class org.osgi.service.dal.FunctionData
getMetadata, getTimestamp

Field Detail

FIELD_SEVERITY

public static final String FIELD_SEVERITY = "severity"

    Represents the severity field name. The field value is available with severity and getSeverity(). The field type is int. The constant can be used as a key to AlarmData(Map).

FIELD_TYPE

public static final String FIELD_TYPE = "type"

    Represents the type field name. The field value is available with type and getType(). The field type is int. The constant can be used as a key to AlarmData(Map).
Interface MultiLevelSensor

```
public static final int TYPE_SMOKE = 1
    The alarm type indicates that smoke is detected.

public static final int TYPE_HEAT = 2
    The alarm type indicates that temperature is too high.

public static final int TYPE_COLD = 3
    The alarm type indicates that temperature is too low.

public static final int TYPE_GAS_CO2 = 4
    The alarm type indicates that carbon dioxide is detected.

public static final int TYPE_GAS_CO = 5
    The alarm type indicates that carbon monoxide is detected.

public static final int TYPE_WATER = 6
    The alarm type indicates that water leak is detected.

public static final int TYPE_POWER_FAIL = 7
    The alarm type indicates a power cut.

public static final int TYPE_HW_FAIL = 8
    The alarm type indicates that there is hardware failure.
```
**Interface MultiLevelSensor**

**TYPE_SW_FAIL**

```java
public static final int TYPE_SW_FAIL = 9
```

The alarm type indicates that there is software failure.

**SEVERITY_NONE**

```java
public static final int SEVERITY_NONE = 0
```

The severity constant indicates that there is no severity rating for this alarm.

**SEVERITY_LOW**

```java
public static final int SEVERITY_LOW = 1
```

The severity rating indicates that there is an alarm with lowest priority.

**SEVERITY_MEDIUM**

```java
public static final int SEVERITY_MEDIUM = 2
```

The severity rating indicates that there is an alarm with medium priority. The severity priority is higher than

```java
SEVERITY_LOW
```

and lower than

```java
SEVERITY_HIGH
```

**SEVERITY_HIGH**

```java
public static final int SEVERITY_HIGH = 3
```

The severity rating indicates that there is an alarm with high priority. The severity priority is higher than

```java
SEVERITY_MEDIUM
```

and lower than

```java
SEVERITY_URGENT
```

**SEVERITY_URGENT**

```java
public static final int SEVERITY_URGENT = 4
```

The severity rating indicates that there an urgent alarm. That severity has highest priority.

**severity**

```java
public final int severity
```

Represents the alarm severity. The field is accessible with `getSeverity()` getter. The vendor can define own alarm severity ratings with negative values.

**type**

```java
public final int type
```
Represents the alarm type. The field is accessible with `getType()` getter. The vendor can define own alarm types with negative values.

### Constructor Detail

#### AlarmData

```java
public AlarmData(Map fields)
```

Constructs new `AlarmData` instance with the specified field values. The map keys must match to the field names. The map values will be assigned to the appropriate class fields. For example, the maps can be: `{"severity"=Integer(1)...}`. That map will initialize the `FIELD_SEVERITY` field with 1. If severity is missing, `SEVERITY_NONE` is used.

**Parameters:**
- `fields` - Contains the new `AlarmData` instance field values.

**Throws:**
- `ClassCastException` - If the field value types are not expected.
- `IllegalArgumentException` - If the alarm type is missing.
- `NullPointerException` - If the fields map is null.

#### AlarmData

```java
public AlarmData(long timestamp,
                 Map metadata,
                 int severity,
                 int type)
```

Constructs new `AlarmData` instance with the specified arguments.

**Parameters:**
- `timestamp` - The alarm data timestamp.
- `metadata` - The alarm data metadata.
- `severity` - The alarm data severity.
- `type` - The alarm data type.

### Method Detail

#### getType

```java
public int getType()
```

Returns the alarm type. The type can be one of the predefined:

- `TYPE_SMOKE`
- `TYPE_HEAT`
- `TYPE_COLD`
- `TYPE_GAS_CO`
- `TYPE_GAS_CO2`
- `TYPE_WATER`
- `TYPE_POWER_FAIL`
- `TYPE_HW_FAIL`
- `TYPE_SW_FAIL`
Interface MultiLevelSensor

The vendor can define own alarm types with negative values.

**Returns:**
The alarm type.

---

getSeverity

```java
public int getSeverity()
```

Returns the alarm severity.

**Returns:**
The alarm severity.

---

equals

```java
public boolean equals(Object other)
```

Two AlarmData instances are equal if they contain equal metadata, timestamp, type and severity.

**Overrides:**
equals in class org.osgi.service.dal.FunctionData

**Parameters:**
other - The object to compare this data.

**Returns:**
true if this object is equivalent to the specified one.

**See Also:**
org.osgi.service.dal.FunctionData.equals(java.lang.Object)

---

hashCode

```java
public int hashCode()
```

Returns the hash code for this AlarmData object. The hash code is a sum of org.osgi.service.dal.FunctionData.hashCode(), the alarm severity and the alarm type.

**Overrides:**
hashCode in class org.osgi.service.dal.FunctionData

**Returns:**
The hash code of this AlarmData object.

**See Also:**
org.osgi.service.dal.FunctionData.hashCode()

---

compareTo

```java
public int compareTo(Object o)
```

Compares this AlarmData instance with the given argument. The argument can be:

- AlarmData - the method returns -1 if metadata, timestamp, type or severity are not equivalent. 0 if all fields are equivalent. 1 if all fields are equivalent and this instance severity is greater than the severity of the specified argument.
- **Map** - the map must be built according the rules of `AlarmData(Map)`. Metadata, timestamp, type and severity are compared according to `AlarmData` argument rules.

**Specified by:**
`compareTo` in interface `Comparable`

**Parameters:**
- `o` - An argument to be compared.

**Returns:**
- `-1`, `0` or `1` depending on the comparison rules.

**Throws:**
- `ClassCastException` - If the method is called with `Map` and the field value types are not expected.
- `IllegalArgumentException` - If the method is called with `Map` and the alarm type is missing.
- `NullPointerException` - If the argument is `null`.

**See Also:**
`Comparable.compareTo(java.lang.Object)`
Class BooleanData

org.osgi.service.dal.functions.data

java.lang.Object
  org.osgi.service.dal.FunctionData
    org.osgi.service.dal.functions.data.BooleanData

All Implemented Interfaces:
  Comparable

public class BooleanData
extends org.osgi.service.dal.FunctionData

Function boolean data wrapper. It can contain a boolean value, timestamp and additional metadata. It doesn't support measurement unit.

See Also:
  BooleanControl, BooleanSensor, org.osgi.service.dal.FunctionData

Field Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD_VALUE</td>
<td>static String</td>
<td>51</td>
</tr>
</tbody>
</table>
  Represents the value field name. |
| value    | boolean            | 51   |
  Represents the boolean value. |

Fields inherited from class org.osgi.service.dal.FunctionData

FIELD_METADATA, FIELD_TIMESTAMP, META_INFO_DESCRIPTION, metadata, timestamp

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BooleanData(Map fields)</td>
<td>51</td>
</tr>
</tbody>
</table>
  Constructs new BooleanData instance with the specified field values. |
| BooleanData(long timestamp, Map metadata, boolean value) | 51   |
  Constructs new BooleanData instance with the specified arguments. |

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>compareTo(Object o)</td>
<td>52</td>
</tr>
</tbody>
</table>
  Compares this BooleanData instance with the given argument. |
| equals(Object other)       | 52   |
  Two BooleanData instances are equal if they contain equal metadata, timestamp and boolean value. |
| getValue()                 | 51   |
  Returns BooleanData value. |
| hashCode()                 | 52   |
  Returns the hash code for this BooleanData object. |

Methods inherited from class org.osgi.service.dal.FunctionData

getMetadata, getTimestamp
Field Detail

FIELD_VALUE

public static final String FIELD_VALUE = "value"

Represents the value field name. The field value is available with value and getValue(). The field type is boolean. The constant can be used as a key to BooleanData(Map).

value

public final boolean value

Represents the boolean value. The field is accessible with getValue() getter.

Constructor Detail

BooleanData

public BooleanData(Map fields)

Constructs new BooleanData instance with the specified field values. The map keys must match to the field names. The map values will be assigned to the appropriate class fields. For example, the maps can be: {"value"=Boolean(true)...}. That map will initialize the FIELD_VALUE field with true.

Parameters:
fields - Contains the new BooleanData instance field values.

Throws:
ClassCastException - If the field value types are not expected.
IllegalArgumentException - If the value is missing.
NullPointerException - If the fields map is null.

BooleanData

public BooleanData(long timestamp,
        Map metadata,
        boolean value)

Constructs new BooleanData instance with the specified arguments.

Parameters:
timestamp - The boolean data timestamp.
metadata - The boolean data metadata.
value - The boolean value.

Method Detail

getValue

public boolean getValue()
equals

public boolean equals(Object other)

Two BooleanData instances are equal if they contain equal metadata, timestamp and boolean value.

Overrides:
equals in class org.osgi.service.dal.FunctionData
Parameters:
other - The object to compare this data.
Returns:
true if this object is equivalent to the specified one.
See Also:
org.osgi.service.dal.FunctionData.equals(java.lang.Object)

hashCode

public int hashCode()

Returns the hash code for this BooleanData object. The hash code is a sum of
org.osgi.service.dal.FunctionData.hashCode() and Boolean.hashCode(), where
Boolean.hashCode() represents the boolean value hash code.

Overrides:
hashCode in class org.osgi.service.dal.FunctionData
Returns:
The hash code of this BooleanData object.
See Also:
org.osgi.service.dal.FunctionData.hashCode()

compareTo

public int compareTo(Object o)

Compares this BooleanData instance with the given argument. The argument can be:

- Boolean - the method returns 0 if this instance contains equivalent boolean value. -1 if this
  instance contains false and the argument is true. 1 if this instance contains true and the
  argument is false.
- BooleanData - the method returns -1 if metadata or timestamp are not equivalent. Otherwise, the
  boolean value is compared with the same rules as Boolean argument.
- Map - the map must be built according the rules of BooleanData(Map). Metadata, timestamp and
  value are compared according BooleanData and Boolean argument rules.

Specified by:
compareTo in interface Comparable
Parameters:
o - An argument to be compared.
Interface MultiLevelSwitch

Returns:
-1, 0 or 1 depending on the comparison rules.

Throws:
- ClassCastException - If the method is called with Map and field value types are not expected.
- IllegalArgumentException - If the method is called with Map and the value is missing.
- NullPointerException - If the argument is null.

See Also:
Comparable.compareTo(java.lang.Object)
Class KeypadData

org.osgi.service.dal.functions.data

java.lang.Object
  └ org.osgi.service.dal.FunctionData
      └ org.osgi.service.dal.functions.data.KeypadData

All Implemented Interfaces:
  Comparable

public class KeypadData
extends org.osgi.service.dal.FunctionData

Represents a keypad event data that is collected when a change with some key from device keypad has occurred. The key code is mapped to FunctionData value.

See Also:
  Keypad, org.osgi.service.dal.FunctionData

Field Summary

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int EVENT_TYPE_PRESSED</td>
<td>Represents a keypad event type for a key pressed.</td>
</tr>
<tr>
<td>static int EVENT_TYPE_PRESSED_DOUBLE</td>
<td>Represents a keypad event type for a double key pressed.</td>
</tr>
<tr>
<td>static int EVENT_TYPE_PRESSED_DOUBLE_LONG</td>
<td>Represents a keypad event type for a double and long key pressed.</td>
</tr>
<tr>
<td>static int EVENT_TYPE_PRESSED_LONG</td>
<td>Represents a keypad event type for a long key pressed.</td>
</tr>
<tr>
<td>static int EVENT_TYPE_RELEASED</td>
<td>Represents a keypad event type for a key released.</td>
</tr>
<tr>
<td>static int EVENT_TYPE_UNKNOWN</td>
<td>Represents an unknown keypad event type.</td>
</tr>
<tr>
<td>int eventType</td>
<td>Represents the keypad event type.</td>
</tr>
<tr>
<td>static String FIELD_EVENT_TYPE</td>
<td>Represents the event type field name.</td>
</tr>
<tr>
<td>static String FIELD_KEY_CODE</td>
<td>Represents the key code field name.</td>
</tr>
<tr>
<td>static String FIELD_KEY_NAME</td>
<td>Represents the key name field name.</td>
</tr>
<tr>
<td>int keyCode</td>
<td>Represents the key code.</td>
</tr>
<tr>
<td>String keyName</td>
<td>Represents the key name, if it's available.</td>
</tr>
</tbody>
</table>

Fields inherited from class org.osgi.service.dal.FunctionData
FIELD_METADATA, FIELD_TIMESTAMP, META_INFO_DESCRIPTION, metadata, timestamp
## Constructor Summary

**KeypadData** (Map fields)  
Constructs new KeypadData instance with the specified field values.

**KeypadData** (long timestamp, Map metadata, int eventType, int keyCode, String keyName)  
Constructs new KeypadData instance with the specified arguments.

## Method Summary

**int compareTo(Object o)**  
Compares this KeypadData instance with the given argument.

**boolean equals(Object other)**  
Two KeypadData instances are equal if they contain equal metadata, timestamp, event type, key code and key name.

**int getEventType()**  
Returns the event type.

**int getKeyCode()**  
The code of the key.

**String getKeyName()**  
Represents a human readable name of the corresponding key code.

**int hashCode()**  
Returns the hash code for this KeypadData object.

## Field Detail

### FIELD_KEY_NAME

public static final String **FIELD_KEY_NAME** = "keyName"

Represents the key name field name. The field value is available with **keyName** and **getKeyName()**. The field type is **String**. The constant can be used as a key to **KeypadData(Map)**.

### FIELD_EVENT_TYPE

public static final String **FIELD_EVENT_TYPE** = "eventType"

Represents the event type field name. The field value is available with **eventType** and **getEventType()**. The field type is **int**. The constant can be used as a key to **KeypadData(Map)**.

### FIELD_KEY_CODE

public static final String **FIELD_KEY_CODE** = "keyCode"

Represents the key code field name. The field value is available with **keyCode** and **getKeyCode()**. The field type is **int**. The constant can be used as a key to **KeypadData(Map)**.
EVENT_TYPE_UNKNOWN

public static final int EVENT_TYPE_UNKNOWN = 0

Represents an unknown keypad event type.

EVENT_TYPE_PRESSED

public static final int EVENT_TYPE_PRESSED = 1

Represents a keypad event type for a key pressed.

EVENT_TYPE_PRESSED_LONG

public static final int EVENT_TYPE_PRESSED_LONG = 2

Represents a keypad event type for a long key pressed.

EVENT_TYPE_PRESSED_DOUBLE

public static final int EVENT_TYPE_PRESSED_DOUBLE = 3

Represents a keypad event type for a double key pressed.

EVENT_TYPE_PRESSED_DOUBLE_LONG

public static final int EVENT_TYPE_PRESSED_DOUBLE_LONG = 4

Represents a keypad event type for a double and long key pressed.

EVENT_TYPE_RELEASED

public static final int EVENT_TYPE_RELEASED = 5

Represents a keypad event type for a key released.

eventType

public final int eventType

Represents the keypad event type. The vendor can define own event types with negative values. The field is accessible with getEventType() getter.

keyName

public final String keyName

Represents the key name, if it's available. The field is accessible with getKeyName() getter.
keyCode

public final int keyCode

Represents the key code. This field is mandatory and it holds the semantics(meaning) of the key. The field is accessible with getKeyCode() getter.

Constructor Detail

KeypadData

public KeypadData(Map fields)

Constructs new KeypadData instance with the specified field values. The map keys must match to the field names. The map values will be assigned to the appropriate class fields. For example, the maps can be: {"eventType"=Integer(1)...}. That map will initialize the FIELD_EVENT_TYPE field with 1.

FIELD_EVENT_TYPE field value type must be Integer. FIELD_KEY_CODE field value type must be Integer. FIELD_KEY_NAME field value type must be String.

Parameters:
fields - Contains the new KeypadData instance field values.

Throws:
ClassCastException - If the field value types are not expected.
IllegalArgumentException - If the event type or key code is missing.
NullPointerException - If the fields map is null.

KeypadData

public KeypadData(long timestamp,
Map metadata,
int eventType,
int keyCode,
String keyName)

Constructs new KeypadData instance with the specified arguments.

Parameters:
timestamp - The data timestamp.
metadata - The data metadata.
eventType - The data event type.
keyCode - The data key code.
keyName - The data key name.

Method Detail

getEventType

public int getEventType()

Returns the event type. The vendor can define own event types with negative values.

Returns:
The event type.
getKeyCode

public int getKeyCode()

The code of the key. This field is mandatory and it holds the semantics(meaning) of the key.

Returns:
The key code.

gKeyName

public String getKeyName()

Represents a human readable name of the corresponding key code. This field is optional and sometimes it could be missed(might be null).

Returns:
A string with the name of the key or null if not specified.

equals

public boolean equals(Object other)

Two KeypadData instances are equal if they contain equal metadata, timestamp, event type, key code and key name.

Overrides:
equals in class org.osgi.service.dal.FunctionData
Parameters:
other • The object to compare this data.
Returns:
true if this object is equivalent to the specified one.
See Also:
org.osgi.service.dal.FunctionData.equals(java.lang.Object)

hashCode

public int hashCode()

Returns the hash code for this KeypadData object. The hash code is a sum of org.osgi.service.dal.FunctionData.hashCode(), String.hashCode(), event type and key code, where String.hashCode() represents the key name hash code if available.

Overrides:
hashCode in class org.osgi.service.dal.FunctionData
Returns:
The hash code of this LevelData object.
See Also:
org.osgi.service.dal.FunctionData.hashCode()
Compares this KeypadData instance with the given argument. The argument can be:

- **KeypadData** - the method returns -1 if metadata, timestamp, event type, key code or key name are not equivalent. 0 if all fields are equivalent.
- **Map** - the map must be built according the rules of KeypadData(Map). Metadata, timestamp, event type, key code and key name are compared according KeypadData argument rules.

Specified by:
```
compareTo in interface Comparable
```

Parameters:
- `o` - An argument to be compared.

Returns:
-1 or 0 depending on the comparison rules.

Throws:
- `ClassCastException` - If the method is called with Map and the field value types are not expected.
- `IllegalArgumentException` - If the method is called with Map and the event type or key code is missing.
- `NullPointerException` - If the argument is null.

See Also:
```
Comparable.compareTo(java.lang.Object)
```

```
public class LevelData
extends org.osgi.service.dal.FunctionData

Function level data wrapper. It supports all properties defined in FunctionData.

See Also:
MultiLevelControl, MultiLevelSensor, Meter, org.osgi.service.dal.FunctionData

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD_LEVEL</td>
<td>String</td>
<td>Represents the level field name.</td>
<td>61</td>
</tr>
<tr>
<td>FIELD_UNIT</td>
<td>String</td>
<td>Represents the unit field name.</td>
<td>61</td>
</tr>
<tr>
<td>level</td>
<td>BigDecimal</td>
<td>Represents the current level.</td>
<td>61</td>
</tr>
<tr>
<td>unit</td>
<td>String</td>
<td>Represent the unit as it's defined in org.osgi.service.dal.PropertyMetadata.UNITS.</td>
<td>61</td>
</tr>
</tbody>
</table>

Fields inherited from class org.osgi.service.dal.FunctionData
FIELD_METADATA, FIELD_TIMESTAMP, META_INFO_DESCRIPTION, metadata, timestamp

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LevelData(Map fields)</td>
<td>Constructs new LevelData instance with the specified field values.</td>
<td>61</td>
</tr>
<tr>
<td>LevelData(long timestamp, Map metadata, String unit, BigDecimal level)</td>
<td>Constructs new LevelData instance with the specified arguments.</td>
<td>62</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>compareTo</td>
<td>Compares this LevelData instance with the given argument.</td>
<td>63</td>
</tr>
<tr>
<td>equals</td>
<td>Two LevelData instances are equal if they contain equal metadata, timestamp, unit and level.</td>
<td>62</td>
</tr>
<tr>
<td>getLevel</td>
<td>Returns LevelData value.</td>
<td>62</td>
</tr>
<tr>
<td>getUnit</td>
<td>Returns LevelData unit as it's specified in org.osgi.service.dal.PropertyMetadata.UNITS or null if the unit is missing.</td>
<td>62</td>
</tr>
</tbody>
</table>
Interface OpenClose

```
<table>
<thead>
<tr>
<th>int hashCode()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the hash code for this LevelData object.</td>
</tr>
<tr>
<td>----------------</td>
</tr>
</tbody>
</table>
```

Methods inherited from class org.osgi.service.dal.FunctionData

getMetadata, getTimestamp

Field Detail

FIELD_LEVEL

```
public static final String FIELD_LEVEL = "level"
```

Represents the level field name. The field value is available with level and getLevel(). The field type is BigDecimal. The constant can be used as a key to LevelData(Map).

FIELD_UNIT

```
public static final String FIELD_UNIT = "unit"
```

Represents the unit field name. The field value is available with unit and getUnit(). The field type is String. The constant can be used as a key to LevelData(Map).

unit

```
public final String unit
```

Represent the unit as it's defined in org.osgi.service.dal.PropertyMetadata.UNITS. The field is optional. The field is accessible with getUnit() getter.

level

```
public final BigDecimal level
```

Represents the current level. It's mandatory field. The field is accessible with getLevel() getter.

Constructor Detail

LevelData

```
public LevelData(Map fields)
```

Constructs new LevelData instance with the specified field values. The map keys must match to the field names. The map values will be assigned to the appropriate class fields. For example, the maps can be: {"level"=BigDecimal(1)...}. That map will initialize the FIELD_LEVEL field with 1.

FIELD_UNIT field value type must be String, FIELD_LEVEL field value type must be BigDecimal.

Parameters:

fields - Contains the new LevelData instance field values.

Throws:

ClassCastException - If the field value types are not expected.

IllegalArgumentException - If the level is missing.
NullPointerException - If the fields map is null.

**LevelData**

```java
public LevelData(long timestamp,
                 Map metadata,
                 String unit,
                 BigDecimal level)
```

Constructs new LevelData instance with the specified arguments.

**Parameters:**
- timestamp - The data timestamp.
- metadata - The data metadata.
- unit - The data unit.
- level - The level value.

**Method Detail**

**getLevel**

```java
public BigDecimal getLevel()
```

Returns LevelData value. The value type is BigDecimal instead of double to guarantee value accuracy.

**Returns:**
- The LevelData value.

**getUnit**

```java
public String getUnit()
```

Returns LevelData unit as it's specified in org.osgi.service.dal.PropertyMetadata.UNITS or null if the unit is missing.

**Returns:**
- The value unit or null if the unit is missing.

**equals**

```java
public boolean equals(Object other)
```

Two LevelData instances are equal if they contain equal metadata, timestamp, unit and level.

**Overrides:**
- equals in class org.osgi.service.dal.FunctionData

**Parameters:**
- other - The object to compare this data.

**Returns:**
- true if this object is equivalent to the specified one.

**See Also:**
- org.osgi.service.dal.FunctionData.equals(java.lang.Object)
Interface OpenClose

hashCode

public int hashCode()

Returns the hash code for this LevelData object. The hash code is a sum of 
org.osgi.service.dal.FunctionData.hashCode(), String.hashCode() and BigDecimal.hashCode(), 
where String.hashCode() represents the unit hash code and BigDecimal.hashCode() represents the 
level hash code.

Overrides:
hashCode in class org.osgi.service.dal.FunctionData

Returns:
The hash code of this LevelData object.

See Also:
org.osgi.service.dal.FunctionData.hashCode()

compareTo

public int compareTo(Object o)

Compares this LevelData instance with the given argument. The argument can be:

- BigDecimal - the method returns the result of BigDecimal.compareTo(Object) for this instance 
  level and the specified argument.
- LevelData - the method returns -1 if metadata, timestamp or unit are not equivalent. Otherwise, 
  the level is compared with the same rules as BigDecimal argument.
- Map - the map must be built according the rules of LevelData(Map). Metadata, timestamp, unit and 
  level are compared according BigDecimal and LevelData argument rules.

Specified by:
compareTo in interface Comparable

Parameters:
o - An argument to be compared.

Returns:
-1, 0 or 1 depending on the comparison rules.

Throws:
ClassCastException - If the method is called with Map and the field value types are not expected. 
IllegalArgumentException - If the method is called with Map and the level is missing. 
NullPointerException - If the argument is null.

See Also:
Comparable.compareTo(java.lang.Object)
8 Considered Alternatives

Currently, there are no alternatives.

9 Security Considerations

Currently, the security is covered by OSGi Device Abstraction Layer.

10 Document Support

10.1 References


10.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Evgeni Grigorov</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>ProSyst Software</td>
</tr>
<tr>
<td>Address</td>
<td>Aachenerstr. 222, 50935 Cologne, Germany</td>
</tr>
<tr>
<td>Voice</td>
<td>+49 221 6604 501</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:e.grigorov@prosyst.com">e.grigorov@prosyst.com</a></td>
</tr>
</tbody>
</table>
10.3 End of Document
Abstract

The component model defined by Declarative Services is using a method based approach for injecting referenced services into the component. Compared to other component models this requires the developer to write the same boiler plate code for each and every reference. This RFC aims to provide a technical design to add field injection to Declarative Services.

This RFC focuses on field injection for Declarative Services.
0  Document Information

0.1  License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

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

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

The OSGi Alliance is willing to receive input, suggestions and other feedback ("Feedback") on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable, worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise
I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0 Document Information .................................................................................................................................. 2
   0.1 License .................................................................................................................................................. 2
   0.2 Trademarks ........................................................................................................................................... 3
   0.3 Feedback ................................................................................................................................................ 3
   0.4 Table of Contents .................................................................................................................................... 3
   0.5 Terminology and Document Conventions ............................................................................................. 4
   0.6 Revision History ...................................................................................................................................... 4

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

2 Application Domain ..................................................................................................................................... 5
   2.1 Terminology + Abbreviations .................................................................................................................. 6

3 Problem Description ..................................................................................................................................... 6

4 Requirements ................................................................................................................................................ 7

5 Technical Solution ....................................................................................................................................... 7
   5.1 Supported Field Types ............................................................................................................................ 7
   5.2 Field Injection Strategies ........................................................................................................................ 8
      5.2.1 Static References ............................................................................................................................... 8
      5.2.2 Unary Dynamic References .............................................................................................................. 8
      5.2.3 Multiple Dynamic References ......................................................................................................... 9
      5.2.4 Multiple Dynamic References – Update Strategy ........................................................................... 9
      5.2.5 Multiple Dynamic References – Replace Strategy ........................................................................ 9
   5.3 Aggregate Types ....................................................................................................................................... 9
      5.3.1 Replace Strategy ................................................................................................................................. 9
      5.3.2 Update Strategy ................................................................................................................................ 10
0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

Source code is shown in this typeface.

0.6 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>04.07.14</td>
<td>Initial proposal – replace strategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Carsten Ziegeler (Adobe Systems Incorporated)</td>
</tr>
<tr>
<td>Update</td>
<td>17.07.14</td>
<td>Update</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Different strategies – event and replace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Avoid type evaluation at runtime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Carsten Ziegeler (Adobe Systems Incorporated)</td>
</tr>
</tbody>
</table>
1 Introduction

The component model defined by Declarative Services is using a method based approach for injecting referenced services into the component. Compared to other component models this requires the developer to write the same boiler plate code for each and every reference. This RFC aims to provide a technical design to add field injection to Declarative Services..

This RFC focuses on field injection for Declarative Services.

2 Application Domain

Declarative Services (chapter 112 in the OSGi specifications) defines a POJO programming model for OSGi services. While RFC 190 and RFC 208 aim at making component development with DS easier and try to reduce the amount of code to write, DS is using an event strategy based on method injection and therefore still requires the developer to implement bind/unbind/update methods for each and every reference. In most cases the code of these methods is always the same and usually simply updates a field in the component holding the referenced service. While the method provides a notification mechanism, too, this is rarely used.

The Apache Felix SCR Annotations and tooling based on these annotations provide an annotation to be used on a field holding a unary reference. The tooling generates byte code for a class holding such an annotation and adds the bind/unbind methods automatically, reducing the boiler plate code to be written by a component developer.
In other component models, like Apache Felix iPojo, CDI or the Spring Framework, field injection is very popular and field injection missing in DS has always been a larger criticism against DS.

DS supports four reference cardinality modes. In addition to supporting more than one reference, a reference can be optional or mandatory. That is, a reference can be satisfied with zero or one bound service. In addition, RFC 190 introduces the minimum cardinality property which allows to raise the specified minimum value to a higher number.

2.1 Terminology + Abbreviations

DS  Declarative Services

POJO  Plain old Java Object; term use for objects not implementing and framework specific plumbing such as Servlet API, Spring API, or OSGi API.

SCR  Service Components Runtime; generally the implementation of the Declarative Services Specification; also the name of the Apache Felix implementation (Apache Felix SCR).

3 Problem Description

The current DS component model for handling references supports two different ways, the lookup strategy and the event strategy. When using the lookup strategy, a service is lookup through the `ComponentContext` each time it is used. The event strategy is based on implementing `bind/unbind/update` methods. The model describes when and in which order these methods are invoked. This depends on the cardinality of the reference (unary or multiple), whether the reference is mandatory and whether the reference is dynamic or static.

Field injection can be added to the model in two ways:

- By just defining a new annotation which is processed by tooling and the tooling enhances the class with corresponding method implementations. This is the approach the Apache Felix SCR tooling has taken and requires no changes to the DS specification.

- Adding field annotation as a first class citizen to the component model. This requires changes/additions to the DS spec, the XML schema, and the implementation. In addition an annotation needs to be defined. The benefit of this solution is that it does not depend on any specific tooling.

In contrast to method based injection, field injection moves (at least part of) the burden of proper synchronizing the access to the field to the implementation of field injection (either the DS implementation or the generated byte code). With method based injection, the burden lies solely on the component developer. Therefore field injection should make the life of the developer easier within the limitations of field injection.
4 Requirements

FID001 – The solution MUST provide a way to define field injection when developing DS components.

FID002 – The solution MUST support the same functionality as the reference handling through methods.

FID003 – The solution SHOULD outline the implications for the component developer with respect to thread safety concerns for accessing the value of the injected field.

FID004 – The solution SHOULD not be tied to Java 5+. It should be usable with lower Java versions.

5 Technical Solution

The technical solution proposes changes in DS, enhancing the XML schema and a new annotation for field injection.

As field injection provides the same functionality as method injection, most of the concepts from method injection can be reused as is, this includes defining the policy, the policy-option and the target filter. The solution for field injection provides the same options for the cardinality of a reference as method injection (0..1, 1..1, 0..n, 1..n) including raising the minimum cardinality as outlined by RFC190. The policy can either be dynamic or static and the policy option is either greedy or reluctant.

5.1 Supported Field Types

If a field references a service of type \textit{SE} and \textit{IN} is a type that is assignable from \textit{SE}, the following types are supported for a field of cardinality unary:

- \textit{IN}
- \texttt{org.osgi.framework.ServiceReference}
- \texttt{org.osgi.framework.ServiceObjects}
- \texttt{java.util.Map} for injecting the service properties (Map\langle String, Object\rangle)
- \texttt{java.util.Map.Entry} - the key of the entry is a map containing the service properties (Map\langle String, Object\rangle) and the value is the service (IN).

The \texttt{java.util.Map} containing the service properties additionally implements \texttt{Comparable}. The \texttt{compareTo()} method compares map objects in the same way \texttt{ServiceReference.compareTo} does based on service ranking and service id. The provided \texttt{java.util.Map.Entry} implements \texttt{Comparable} in the same way.
For a field reference of cardinality multiple different aggregate types of one of the unary types as defined above are supported:

- `java.util.Collection`
- `java.util.List`
- any type assignable to `java.util.Collection` if the component provides the implementation and the update strategy is used for this field (see below)

Other field types are not supported. If a component is using an unsupported type, the component is not activated and the error situation must be logged. Tooling might already detect the situation at build time and can issue an error to the developer.

At runtime, the DS implementation reads the component XML (see 5.4) and therefore gets the cardinality of the field and the type of the reference SE. If the cardinality is unary, the DS implementation can detect the type of the field through reflection. For the rare case that the referenced service is one of `ServiceReference`, `ServiceObjects`, `java.util.Map` or `java.util.Map.Entry` and the same type is used for the field, only the service itself will be injected into the field.

For cardinality multiple the aggregate type is always a subtype of `java.util.Collection`, the XML contains the information about the aggregated type.

## 5.2 Field Injection Strategies

### 5.2.1 Static References

For fields holding a static reference, either unary or multiple, the value of the field is set once before the component activator is called and never touched again by DS. The usage of the provided value is therefore thread-safe. If a change in the referenced services occurs, the component instance is discarded and a new instance is created. For static references only this strategy, named the replace strategy, is allowed and therefore for static references of cardinality multiple, DS will always provide the implementation of the collection. If a different strategy is specified in the XML, the component is not activated and this error must be logged.

### 5.2.2 Unary Dynamic References

For fields of cardinality unary the replace strategy is always used. With this strategy the value of the field is replaced whenever changes regarding the referenced service occur. The field is set by DS in the same way and order as DS would call the methods for method injection:

- If the reference becomes satisfied, the field is set to a value according to the used type
- If the bound service is replaced (see 112.5.10), the field is set to the new value.
- If a reference becomes unsatisfied, the field is set to `null`.
- If the service properties of a bound service are modified, the value of the field is updated if it contains the service properties (`java.util.Map` or `java.util.Map.Entry`). In other cases the field does not need to be updated as the value of the field does not change.

The field must be declared as volatile. Otherwise other threads than the thread setting the field might never see an update of the field. If a component is using a non-volatile field for injection a dynamic unary reference, the component is not activated. This error must be logged. In addition, the tooling processing an annotated field should already signal an error for this situation.
5.2.3 Multiple Dynamic References

For fields with a dynamic reference of type multiple two different strategies can be used: the replace and the update strategy.

5.2.4 Multiple Dynamic References – Update Strategy

In the case of the update strategy for references of cardinality multiple, the field is set once to the corresponding aggregate implementation (see below) and whenever changes to the set of referenced services occur, the collection is directly modified:

- If a developer is providing an implementation for the aggregate type, this needs to be done as part of the component object construction. If the field does not contain a value after constructing the object, it is set by the DS implementation before the component activator is called. (More about the aggregate implementation in chapter 5.3.)
- For each bound service, Collection.add() is called on the aggregate.
- If a service is unbound, Collection.remove() is called on the aggregate.
- If the service properties of a bound service are modified, Collection.add() followed by Collection.remove() is called if the aggregated type contains the service properties (java.util.Map or java.util.Map.Entry). In other cases, the collection is not modified.

5.2.5 Multiple Dynamic References – Replace Strategy

With the replace strategy, always a new mutable collection is created and set as the value of the field. The field is set by DS in the same way and order as DS would call the methods for method injection:

- Before the component activator is called, the field is initialized with a collection containing the currently bound references. A value set by component code as part of construction the instance will be overwritten.
- When a new service is bound, the field is set to a new collection including the new service.
- When a service is unbound, the field is set to a new collection without that service. If there is no matching service, an empty collection is set as the value.
- If the service properties of a bound service are modified, the field is updated with a new collection if the aggregated type contains the service properties (java.util.Map or java.util.Map.Entry).

The field must be declared as volatile. Otherwise other threads than the thread setting the field might never see an update of the field. If a component is using a non-volatile field for injection in this case, the component is not activated. This error must be logged. In addition, the tooling processing an annotated field should already signal an error for this situation.

5.3 Aggregate Types

5.3.1 Replace Strategy

If the replace strategy is used, the DS implementation will pick the aggregate implementation. The aggregate type of the field must be one of

- java.util.Collection
- java.util.List

Other field types are not supported for the replace strategy. If a component is using a different type for this case, the component is not activated and an error must be logged. Tooling can already detect this error at build time and report it to the developer. The field must not be declared final. If it is declared as final, the component is not activated and an error must be logged. Tooling can already detect this error at build time and report it to the developer.
The collection is based on object identity, mutable and sorted as described by ServiceReference.compareTo().

5.3.2 Update Strategy

If the update strategy is used, the component developer can decide between providing an implementation for the aggregate type or letting the DS implementation choose an implementation. If a developer is providing an implementation for the aggregate type, this needs to be done as part of the component object construction. If the field does not contain a value after constructing the object, it is set by the DS implementation before the component activator is called.

If the DS implementation provides the implementation of the aggregate type, the provided collection is based on object identity, thread safe and can safely be used concurrently. The collection is mutable and sorted as described by ServiceReference.compareTo(). However the collection is not sorted. The type of the field must either be java.util.Collection or java.util.List. If a different type is used, the component is not activated and an error is logged. Tooling can already detect this error at build time. The field must not be declared final. If it is declared as final, the component is not activated and an error must be logged. Tooling can already detect this error at build time and report it to the developer.

If the component developer provides an implementation for the aggregate type, the field needs to be set during construction of the instance. The type of the field can be any type assignable to java.util.Collection. Collection.add and Collection.remove are used on the aggregate type to update the aggregate. A developer should not rely on equals or hashCode of the provided objects to detect which object to remove from the collection. It should rather be checked for identity of the object. The DS implementation ensures to pass the same object to the remove method as it passed to the add method. In addition, a thread safe aggregate implementation must be used or the access needs to be properly synchronized to avoid runtime errors like a concurrent modification exception. The field should be declared as final.

Whether DS provides the aggregate or the component implementation provides it, the DS implementation is always treating the field as if declared final and might cache the field value and therefore will never be aware of any changes to the field value.

5.4 XML Schema

For field injection a new element field-reference is added to the component XML schema with the attributes policy, policy-option, cardinality, scope, target and interface. These attributes have the same values and meaning as those for the reference element. The attribute field contains the name of the field within the component class.

In addition the strategy attribute can either have the value replace or update. If it is not specified, replace is used as the default. For unary references specifying update is considered an error and the component is not activated. This case must be logged.

In the case of references with cardinality multiple, the runtime needs to have information about the aggregated type. The attribute valuetype can be used to specify the type. Allowed values are service, properties, reference, serviceobjects, or tuple. If not specified it defaults to service. For unary references specifying valuetype is considered an error and the component is not activated. This case must be logged.
5.5 Annotation

A new annotation @FieldReference is added that can be used to annotate a field. The attributes policy, policy-option, cardinality, scope, target and service have the same meaning as the equivalents for the @Reference annotation and are mapped in the same way to the counterparts in the XML.

The values of the different XML attributes for the field reference are tried to be deduced by the tooling depending on the type of the annotated field.

If the cardinality is not specified as part of the annotation, the cardinality is detected depending on the type of the field. If the type of the field is one of java.util.Collection, or java.util.List the cardinality defaults to optional multiple (0..n), otherwise it is set to unary mandatory (1..1).

If the field is marked as volatile, the policy defaults to dynamic, otherwise it defaults to static.

If the field is marked as final, the strategy is set to update.

If the type of the field or the aggregated type for collections is not SE (the service type) but a type that is assignable from SE, the annotation attribute service must be set to the service type SE. By default, the type of the field, the aggregated type or the generic type (for ServiceReference, ServiceObjects) are used as the service type.

The value for valuetype in the component XML is deduced by the generic type information of the aggregate and the aggregated type.

5.6 Component Development

Field injection has some implications on the code written by the component developer:

- A field used for field injection must be treated with care by client code. There is no way for the DS implementation to check/ensure whether client code is altering the value of the field or the contents of a collection set to the field. Therefore it's suggested to not change the value of the field or the collection from client code. **For multiple cardinality, the DS implementation is always treating the field as if declared final and might cache the field value and therefore will never be aware of any changes to the field value.**

- Type safety can only be validated up to a certain point when using Java 5+. The DS implementation solely relies on the component XML to provide the correct type information, if a wrong type information is provided, a ClassCastException might occur at runtime. However the annotation tooling should try to check for wrongly used types and report this to the developer. If no generic information is available, the tooling should at least issue a warning.

- Static fields can't be used for field reference. If a component is trying to use such a field for field injection, the component is not activated. This error should be logged. In addition, tooling can already report this as an error at build time.

- Final fields can only be used for a field reference if the component developer provides the aggregate implementation in the case of a reference of cardinality multiple. If a component is trying to use such a field for field injection in other cases, the component is not activated. This error should be logged. In addition, tooling can already report this as an error at build time.

- In the case of the update strategy, the add/remove methods of the aggregate type are used to update the aggregate. A developer should not rely on equals or hashCode or the provided objects to detect which object to remove from the collection. It should rather be checked for identity of the object.

5.7 Examples

Example for unary reference:

```java
@FieldReference(policy=ReferencePolicy.DYNAMIC)
private volatile MyService service;
```
public void doIt() {
    final MyService localService = this.service;
    if (localService != null) {
        // use service
    } else {
        // do something without service
    }
}

Example for multiple reference with replace strategy:

@FieldReference(policy=ReferencePolicy.DYNAMIC)
private volatile List<MyService> serviceList;

public void doItList() {
    final List<MyService> localList = this.serviceList;
    if (!localList.isEmpty()) {
        for (final MyService ms : localList) {
            // do something with ms
        }
    } else {
        // no service available, do something else
    }
}

Example for multiple reference with update strategy, DS provided collection

@FieldReference(policy=ReferencePolicy.DYNAMIC,
                 strategy=ReferenceStrategy.UPDATE)
private List<MyService> serviceList;

public void doItList() {
    for (final MyService ms : serviceList) {
        // do something with ms
    }
}

Example for multiple reference with update strategy, component provided collection

@FieldReference(policy=ReferencePolicy.DYNAMIC,
                 strategy=ReferenceStrategy.UPDATE)
private final List<MyService> serviceList =
    new java.util.concurrent.CopyOnWriteArrayList();

public void doItList() {
    for (final MyService ms : serviceList) {
        // do something with ms
    }
}
5.8 Updates to DS

Section 5.1 introduces a comparable `java.util.Map` for service properties. Whenever a map is based to a method used for method injection, this implementation will be passed, allowing the implementor of the method to easily sort the references based on the provided map.

6 Data Transfer Objects

A DTO for field injection is required which is similar to the reference DTO with the difference that it points to a field instead of listing the different methods. In addition it contains other information like the used strategy.

The `ReferenceDTO` introduced with RFC-190 is enhanced with a field of type String named `field` containing the field name. In order to distinguish between the different reference types, a field of type String named `strategy` is added to `ReferenceDTO` containing one of the following values: `LOOKUP`, `METHOD`, `FIELD_EVENT`, or `FIELD_UPDATE`. If the value is either `FIELD_EVENT_REPLACE` or `FIELD_UPDATE`, the field name is available via `field` and the fields for the methods all return `null`.

7 Javadoc

TODO
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 Byte Code Generation

A similar solution could also be implemented using byte code generation. The byte code generation would generate complex methods dealing with all the cases. However this solution would depend on specific tooling.

8.2 Volatile vs AtomicXXX

In order to keep the spec simple, AtomicXXX as an alternative to making a field volatile are not supported. Both concepts basically provide the same functionality, therefore limiting it to just volatile.

8.3 Support for Collections in Methods

As described in this RFC, the DS implementation does already the heavy work of creating the collections for field injection, support for new method signatures for the bind method could be added to DS:

- protected void bindMyService(Collection<MyService> serviceCollection)

This is not part of this proposal.

9 Security Considerations

No change from the Declarative Services specification as updated through RFC 190.
10 Document Support

10.1 References


10.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Carsten Ziegeler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Adobe Systems Incorporated</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:cziegele@adobe.com">cziegele@adobe.com</a></td>
</tr>
</tbody>
</table>

10.3 Acronyms and Abbreviations

10.4 End of Document
License Grant

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) or (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, sublicensable, 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.

Trademarks

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

Feedback

This specification can be downloaded from the OSGi Alliance web site:

http://www.osgi.org

Comments about this specification can be raised at:

https://www.osgi.org/bugzilla/
# Table of Contents

## 1 Introduction

1.1 Overview of Services .................................................. 15
1.2 Application and Provisioning Support .................................. 18
1.3 Reader Level ................................................................... 19
1.4 Version Information ....................................................... 20
1.5 References .................................................................... 21
1.6 Changes ........................................................................ 21

## 100 Remote Services

100.1 The Fallacies ................................................................. 23
100.2 Remote Service Properties ............................................. 24
100.3 Intents ......................................................................... 28
100.4 General Usage ............................................................. 28
100.5 Configuration Types ..................................................... 29
100.6 Security ...................................................................... 32
100.7 References .................................................................. 33

## 101 Log Service Specification

101.1 Introduction .................................................................. 35
101.2 The Log Service Interface ............................................. 36
101.3 Log Level and Error Severity ......................................... 37
101.4 Log Reader Service ........................................................ 37
101.5 Log Entry Interface ....................................................... 38
101.6 Mapping of Events ....................................................... 38
101.7 Security ...................................................................... 41
101.8 org.osgi.service.log ...................................................... 41

## 102 Http Service Specification

102.1 Introduction ................................................................. 47
102.2 Registering Servlets ..................................................... 48
102.3 Registering Resources ................................................... 50
102.4 Mapping HTTP Requests to Servlet and Resource Registrations ......................................................... 51
102.5 The Default Http Context Object ................................ 52
102.6 Multipurpose Internet Mail Extension (MIME) Types ......................................................... 53
102.7 Authentication ............................................................ 54
102.8 Security .................................................................... 55
102.9 Configuration Properties ............................................. 56
102.10 org.osgi.service.http .................................................. 56
102.11 References .................................................................. 61

## 104 Configuration Admin Service Specification

104.1 Introduction ................................................................. 63
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>104.1</td>
<td>Introduction</td>
<td>63</td>
</tr>
<tr>
<td>104.2</td>
<td>Configuration Targets</td>
<td>65</td>
</tr>
<tr>
<td>104.3</td>
<td>The Persistent Identity</td>
<td>66</td>
</tr>
<tr>
<td>104.4</td>
<td>The Configuration Object</td>
<td>69</td>
</tr>
<tr>
<td>104.5</td>
<td>Managed Service</td>
<td>72</td>
</tr>
<tr>
<td>104.6</td>
<td>Managed Service Factory</td>
<td>76</td>
</tr>
<tr>
<td>104.7</td>
<td>Configuration Admin Service</td>
<td>80</td>
</tr>
<tr>
<td>104.8</td>
<td>Configuration Events</td>
<td>84</td>
</tr>
<tr>
<td>104.9</td>
<td>Configuration Plugin</td>
<td>85</td>
</tr>
<tr>
<td>104.10</td>
<td>Meta Typing</td>
<td>87</td>
</tr>
<tr>
<td>104.11</td>
<td>Security</td>
<td>87</td>
</tr>
<tr>
<td>104.12</td>
<td>org.osgi.service.cm</td>
<td>89</td>
</tr>
<tr>
<td>104.13</td>
<td>Changes</td>
<td>105</td>
</tr>
<tr>
<td>105</td>
<td>Metatype Service Specification</td>
<td>107</td>
</tr>
<tr>
<td>105.1</td>
<td>Introduction</td>
<td>107</td>
</tr>
<tr>
<td>105.2</td>
<td>Attributes Model</td>
<td>108</td>
</tr>
<tr>
<td>105.3</td>
<td>Object Class Definition</td>
<td>109</td>
</tr>
<tr>
<td>105.4</td>
<td>Attribute Definition</td>
<td>109</td>
</tr>
<tr>
<td>105.5</td>
<td>Meta Type Service</td>
<td>110</td>
</tr>
<tr>
<td>105.6</td>
<td>Meta Type Provider Service</td>
<td>112</td>
</tr>
<tr>
<td>105.7</td>
<td>Using the Meta Type Resources</td>
<td>112</td>
</tr>
<tr>
<td>105.8</td>
<td>Object</td>
<td>118</td>
</tr>
<tr>
<td>105.9</td>
<td>XML Schema</td>
<td>119</td>
</tr>
<tr>
<td>105.10</td>
<td>Limitations</td>
<td>121</td>
</tr>
<tr>
<td>105.11</td>
<td>Related Standards</td>
<td>121</td>
</tr>
<tr>
<td>105.12</td>
<td>Security Considerations</td>
<td>121</td>
</tr>
<tr>
<td>105.13</td>
<td>org.osgi.service.metatype</td>
<td>121</td>
</tr>
<tr>
<td>105.14</td>
<td>References</td>
<td>127</td>
</tr>
<tr>
<td>107</td>
<td>User Admin Service Specification</td>
<td>129</td>
</tr>
<tr>
<td>107.1</td>
<td>Introduction</td>
<td>129</td>
</tr>
<tr>
<td>107.2</td>
<td>Authentication</td>
<td>131</td>
</tr>
<tr>
<td>107.3</td>
<td>Authorization</td>
<td>133</td>
</tr>
<tr>
<td>107.4</td>
<td>Repository Maintenance</td>
<td>135</td>
</tr>
<tr>
<td>107.5</td>
<td>User Admin Events</td>
<td>135</td>
</tr>
<tr>
<td>107.6</td>
<td>Security</td>
<td>136</td>
</tr>
<tr>
<td>107.7</td>
<td>Relation to JAAS</td>
<td>137</td>
</tr>
<tr>
<td>107.8</td>
<td>org.osgi.service.useradmin</td>
<td>137</td>
</tr>
<tr>
<td>107.9</td>
<td>References</td>
<td>148</td>
</tr>
<tr>
<td>110</td>
<td>Initial Provisioning Specification</td>
<td>149</td>
</tr>
<tr>
<td>110.1</td>
<td>Introduction</td>
<td>149</td>
</tr>
<tr>
<td>110.2</td>
<td>Procedure</td>
<td>150</td>
</tr>
</tbody>
</table>
# Declarative Services Specification

112.1 Introduction ........................................ 167
112.2 Components .......................................... 170
112.3 References to Services ............................... 173
112.4 Component Description .............................. 179
112.5 Component Life Cycle ................................ 187
112.6 Component Properties ............................... 196
112.7 Deployment ........................................... 197
112.8 Use of the Annotations .............................. 199
112.9 Service Component Runtime ......................... 200
112.10 Security ............................................... 202
112.11 Component Description Schema ..................... 202
112.12 org.osgi.service.component ....................... 205
112.13 org.osgi.service.component.annotations ........... 210
112.14 References .......................................... 216
112.15 Changes ............................................. 216

# Event Admin Service Specification

113.1 Introduction ........................................ 217
113.2 Event Admin Architecture ............................ 218
113.3 The Event ........................................... 219
113.4 Event Handler ........................................ 220
113.5 Event Publisher ....................................... 221
113.6 Specific Events ...................................... 223
113.7 Event Admin Service ................................. 225
113.8 Reliability ............................................ 227
113.9 Inter-operability with Native Applications ......... 227
113.10 Security ............................................... 227
113.11 org.osgi.service.event ......................... 228

# Blueprint Container Specification

121.1 Introduction ........................................ 239
121.2 Managers ............................................ 242
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>124.3</td>
<td>OSGi JMX Management</td>
<td>393</td>
</tr>
<tr>
<td>124.4</td>
<td>MBeans</td>
<td>396</td>
</tr>
<tr>
<td>124.5</td>
<td>Item</td>
<td>396</td>
</tr>
<tr>
<td>124.6</td>
<td>Security</td>
<td>397</td>
</tr>
<tr>
<td>124.7</td>
<td>org.osgi.jmx</td>
<td>397</td>
</tr>
<tr>
<td>124.8</td>
<td>org.osgi.jmx.framework</td>
<td>402</td>
</tr>
<tr>
<td>124.9</td>
<td>org.osgi.jmx.service.cm</td>
<td>425</td>
</tr>
<tr>
<td>124.10</td>
<td>org.osgi.jmx.service.permissionadmin</td>
<td>427</td>
</tr>
<tr>
<td>124.11</td>
<td>org.osgi.jmx.service.provisioning</td>
<td>428</td>
</tr>
<tr>
<td>124.12</td>
<td>org.osgi.jmx.service.useradmin</td>
<td>429</td>
</tr>
<tr>
<td>124.13</td>
<td>org.osgi.jmx.framework.wiring</td>
<td>436</td>
</tr>
<tr>
<td>124.14</td>
<td>References</td>
<td>443</td>
</tr>
<tr>
<td>125</td>
<td>JDBC™ Service Specification</td>
<td>445</td>
</tr>
<tr>
<td>125.1</td>
<td>Introduction</td>
<td>445</td>
</tr>
<tr>
<td>125.2</td>
<td>Database Driver</td>
<td>446</td>
</tr>
<tr>
<td>125.3</td>
<td>Applications</td>
<td>447</td>
</tr>
<tr>
<td>125.4</td>
<td>Security</td>
<td>448</td>
</tr>
<tr>
<td>125.5</td>
<td>org.osgi.service.jdbc</td>
<td>449</td>
</tr>
<tr>
<td>125.6</td>
<td>References</td>
<td>451</td>
</tr>
<tr>
<td>126</td>
<td>JNDI Services Specification</td>
<td>453</td>
</tr>
<tr>
<td>126.1</td>
<td>Introduction</td>
<td>453</td>
</tr>
<tr>
<td>126.2</td>
<td>JNDI Overview</td>
<td>456</td>
</tr>
<tr>
<td>126.3</td>
<td>JNDI Context Manager Service</td>
<td>458</td>
</tr>
<tr>
<td>126.4</td>
<td>JNDI Provider Admin service</td>
<td>461</td>
</tr>
<tr>
<td>126.5</td>
<td>JNDI Providers</td>
<td>461</td>
</tr>
<tr>
<td>126.6</td>
<td>OSGi URL Scheme</td>
<td>464</td>
</tr>
<tr>
<td>126.7</td>
<td>Traditional Client Model</td>
<td>466</td>
</tr>
<tr>
<td>126.8</td>
<td>Security</td>
<td>468</td>
</tr>
<tr>
<td>126.9</td>
<td>org.osgi.service.jndi</td>
<td>469</td>
</tr>
<tr>
<td>126.10</td>
<td>References</td>
<td>471</td>
</tr>
<tr>
<td>127</td>
<td>JPA Service Specification</td>
<td>473</td>
</tr>
<tr>
<td>127.1</td>
<td>Introduction</td>
<td>473</td>
</tr>
<tr>
<td>127.2</td>
<td>JPA Overview</td>
<td>475</td>
</tr>
<tr>
<td>127.3</td>
<td>Bundles with Persistence</td>
<td>479</td>
</tr>
<tr>
<td>127.4</td>
<td>Extending a Persistence Bundle</td>
<td>481</td>
</tr>
<tr>
<td>127.5</td>
<td>JPA Provider</td>
<td>485</td>
</tr>
<tr>
<td>127.6</td>
<td>Static Access</td>
<td>487</td>
</tr>
<tr>
<td>127.7</td>
<td>Security</td>
<td>487</td>
</tr>
<tr>
<td>127.8</td>
<td>org.osgi.service.jpa</td>
<td>488</td>
</tr>
<tr>
<td>127.9</td>
<td>References</td>
<td>488</td>
</tr>
</tbody>
</table>
# Web Applications Specification

128.1 Introduction ........................................................................................................ 489
128.2 Web Container .................................................................................................. 491
128.3 Web Application Bundle .................................................................................. 492
128.4 Web URL Handler ............................................................................................. 496
128.5 Events ............................................................................................................... 499
128.6 Interacting with the OSGi Environment ............................................................. 500
128.7 Security ............................................................................................................ 501
128.8 References ....................................................................................................... 501

# Coordinator Service Specification

130.1 Introduction ....................................................................................................... 503
130.2 Usage ................................................................................................................ 504
130.3 Coordinator Service ........................................................................................... 513
130.4 Security ............................................................................................................ 533
130.5 org.osgi.service.coordinator .......................................................................... 538
130.6 References ....................................................................................................... 539

# Repository Service Specification

132.1 Introduction ....................................................................................................... 531
132.2 Using a Repository ............................................................................................ 532
132.3 Repository ........................................................................................................ 536
132.4 osgi.content Namespace ................................................................................. 536
132.5 XML Repository Format .................................................................................... 537
132.6 XML Repository Schema ................................................................................ 540
132.7 Security ............................................................................................................ 544
132.8 org.osgi.service.repository ............................................................................ 544
132.9 References ....................................................................................................... 550
132.10 Changes ......................................................................................................... 550

# Service Loader Mediator Specification

133.1 Introduction ....................................................................................................... 551
133.2 Java Service Loader API ................................................................................... 553
133.3 Consumers ....................................................................................................... 554
133.4 Service Provider Bundles ................................................................................ 556
133.5 Service Loader Mediator .................................................................................. 558
133.6 osgi.serviceloader Namespace ..................................................................... 561
133.7 Use of the osgi.extender Namespace ............................................................... 561
133.8 Security ............................................................................................................ 562
133.9 org.osgi.service.serviceloader ...................................................................... 562
133.10 References ....................................................................................................... 563

# Subsystem Service Specification

134.1 Introduction ....................................................................................................... 565
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>138.1</td>
<td>Introduction</td>
<td>677</td>
</tr>
<tr>
<td>138.2</td>
<td>Usage</td>
<td>678</td>
</tr>
<tr>
<td>138.3</td>
<td>Async Service</td>
<td>680</td>
</tr>
<tr>
<td>138.4</td>
<td>The Async Mediator</td>
<td>682</td>
</tr>
<tr>
<td>138.5</td>
<td>Fire and Forget invocations</td>
<td>683</td>
</tr>
<tr>
<td>138.6</td>
<td>Delegating to asynchronous implementations</td>
<td>683</td>
</tr>
<tr>
<td>138.7</td>
<td>Security</td>
<td>684</td>
</tr>
<tr>
<td>138.8</td>
<td>org.osgi.service.async</td>
<td>685</td>
</tr>
<tr>
<td>138.9</td>
<td>org.osgi.service.async.delegate</td>
<td>687</td>
</tr>
<tr>
<td>138.10</td>
<td>References</td>
<td>688</td>
</tr>
<tr>
<td>702.1</td>
<td>Introduction</td>
<td>689</td>
</tr>
<tr>
<td>702.2</td>
<td>JAXP</td>
<td>690</td>
</tr>
<tr>
<td>702.3</td>
<td>XML Parser service</td>
<td>691</td>
</tr>
<tr>
<td>702.4</td>
<td>Properties</td>
<td>691</td>
</tr>
<tr>
<td>702.5</td>
<td>Getting a Parser Factory</td>
<td>691</td>
</tr>
<tr>
<td>702.6</td>
<td>Adapting a JAXP Parser to OSGi</td>
<td>692</td>
</tr>
<tr>
<td>702.7</td>
<td>Usage of JAXP</td>
<td>693</td>
</tr>
<tr>
<td>702.8</td>
<td>Security</td>
<td>694</td>
</tr>
<tr>
<td>702.9</td>
<td>org.osgi.util.xml</td>
<td>694</td>
</tr>
<tr>
<td>702.10</td>
<td>References</td>
<td>697</td>
</tr>
<tr>
<td>705.1</td>
<td>Introduction</td>
<td>699</td>
</tr>
<tr>
<td>705.2</td>
<td>Promise</td>
<td>700</td>
</tr>
<tr>
<td>705.3</td>
<td>Deferred</td>
<td>700</td>
</tr>
<tr>
<td>705.4</td>
<td>Callbacks</td>
<td>701</td>
</tr>
<tr>
<td>705.5</td>
<td>Chaining Promises</td>
<td>702</td>
</tr>
<tr>
<td>705.6</td>
<td>Monad</td>
<td>702</td>
</tr>
<tr>
<td>705.7</td>
<td>Functional Interfaces</td>
<td>704</td>
</tr>
<tr>
<td>705.8</td>
<td>Promises Class</td>
<td>704</td>
</tr>
<tr>
<td>705.9</td>
<td>Security</td>
<td>704</td>
</tr>
<tr>
<td>705.10</td>
<td>org.osgi.util.promise</td>
<td>704</td>
</tr>
</tbody>
</table>
1 Introduction

The OSGi Enterprise Expert Group (EEG) is chartered to define the technical requirements and specifications to tailor and extend the OSGi framework to address information technology software infrastructure use cases found in enterprise scenarios.

The EEG technical areas of concern include:

- Scaling, including multi-container and multi-process environments
- Distributed and/or federated service model for:
  - Multiple OSGi frameworks
  - External, heterogeneous systems
- Requirements for extensions to the OSGi publish/find/bind service model
- Enterprise-class life cycle and configuration management
- Integration of established Java EE technology into OSGi

This specification is based on OSGi Core Release 6. The specification combines previously published, as well as new, OSGi services that address the common use cases of enterprise application and application server developers. It serves as a first reference point for the suggested audience when considering the use of OSGi in their environment to fulfill their own needs or to better serve the needs of their customers. This collection of services is taken from the complete set of available specifications and narrowed down to what can be relevant to the enterprise domain.

The services of the Enterprise Specification have been designed to integrate with OSGi and cooperate with each other. None of the listed service specifications is mandatory; all service specifications are optional. However, services provided must follow their specification completely.

It is not suggested, or expected, that an enterprise solution will incorporate support for all listed specifications, instead a customized subset to satisfy the requirements at hand is recommended. A solution can further include other core and compendium services that are not listed as part of the Enterprise Specification. The selection of appropriate services should be driven by requirements and use cases.

The Enterprise Specification includes the recommended specifications for a number of areas. The services of the Enterprise Specification have been designed to guarantee integration with OSGi and cooperation among each other. Together they address use-cases found in the enterprise context and provide a powerful set of tools to build enterprise OSGi deployments.

These Enterprise Specification areas are described in the following sections.

1.1 Overview of Services

1.1.1 Component Models

While the OSGi framework API is relatively simple to use, it is still considered infrastructure that can bleed into the application code of a bundle. This Enterprise Specification therefore provides multiple, interoperable, Dependency Injection based component models. These DI models ensure decoupling of the application code from the OSGi APIs; they provide an OSGi bundle programming model with minimal implementation dependencies and virtually no accidental complexity in the Java code. Both models provide support for handling the life cycle of services, albeit in different ways. These component models are the Declarative Services Specification and the Blueprint Container Specification:
Overview of Services

• **Declarative Services Specification** - The Declarative Services specification provides dependency injection for services. It handles the service life cycle dynamics by notifying the component or managing the component's life cycle. See chapter *Declarative Services Specification* on page 167.

• **Blueprint Container Specification** - The Blueprint Container is derived from the Spring Dynamic Module project. It provides a general DI framework; services are supported by proxying them and damping their life cycle. See chapter *Blueprint Container Specification* on page 239.

1.1.2 Distributed Services

The OSGi framework provides a local service registry for bundles to communicate through service objects, where a service is an object that one bundle registers and another bundle looks up. The Enterprise Specification enhances this model by defining endpoints that represent services hosted in a remote system. It allows for seamless access to remote services within the OSGi framework without changing the service layer. The remote system may or may not be based on OSGi.

The Enterprise Specification includes the specifications of:

• **Remote Services** - The Remote Services specification defines a number of service properties that participating bundles can use to convey information to a distribution provider. The distribution provider creates endpoints that are accessible to remote clients or registers proxies that access services hosted external to the OSGi framework. See chapter *Remote Services* on page 23.

• **Remote Service Admin Specification** - The Remote Services Admin Service Specification defines an API for the distribution provider and discovery of services in a network. A management agent can use this API to provide an actual distribution policy. This management agent can export and import services as well as discovering services in the network. See *Remote Service Admin Service Specification* on page 339.

1.1.3 Web Applications and HTTP Servlets

Current Enterprise Java architectures almost always require support for web technologies in the form of Java Servlets or Web Applications. The Enterprise Specification includes three complementary service specifications in support:

• **Web Application Specification** - The Web Application specification provides support for web applications written to the Servlet 2.5 specification as well as the JSP 2.1 specification. This specification details how web applications packaged as a WAR or as bundles (WABs) can be installed into an OSGi framework, as well as how this application can use OSGi services. See *Web Applications Specification* on page 489.

• **Http Service Specification** - Bundle developers typically need to develop communication and user interface solutions for standard technologies such as HTTP, HTML, XML, and servlets. The Http Service supports two standard techniques for this purpose: registering servlets and registering resources. See *Http Service Specification* on page 47.

• **Http Whiteboard Service Specification** - The [1] Whiteboard Pattern pattern has shown to be a powerful and flexible mechanism for registering customized functionality with a container. The Http Whiteboard Service Specification allows the registration of Servlet, Servlet Filters, Resources and Servlet-related listeners via the Whiteboard pattern, providing an alternative approach to working with servlets to the Http Service Specification. See ??.

1.1.4 Asynchronous Processing and Event models

A number of specifications focus specifically on asynchronous programming and executing of components, as well as the sending and receiving of events.

• **Event Admin Service Specification** - The Event Admin service provides an inter-bundle communication mechanism. It is based on a event publish and subscribe model, popular in many message based systems. See *Event Admin Service Specification* on page 217.
• **Asynchronous Services** - Asynchronous processing can be the key to scalability for large enterprise applications, especially under heavy load. OSGi Services have traditionally followed the Java interface-based design which by default provides synchronous semantics. The Asynchronous Services specification adds an asynchronous programming model to new and existing OSGi services. See [Asynchronous Service Specification](#) on page 677.

• **Promises** - Many JavaScript applications use Promises-based APIs to facilitate asynchronous processing of a workflow in which executions are time consuming or subject to waiting for I/O operations. The OSGi Promises specification defines a Promises API for use in OSGi applications. The Promises API is used by other specifications, such as the Asynchronous Services specification, but can also be used independently.

### 1.1.5 Management and Configuration services

Support for managing the servers and their applications is essential to all enterprise systems. The Enterprise Specification includes several services addressing the need to manage the framework from the outside as well as configuring individual bundles and applications from within the OSGi framework.

• **JMX™ Management Model Specification** - The Java Management Extensions (JMX) is the standard API specification for providing a management interface to Java SE and Java EE applications. The JMX Management Model specification provides an MBean interface adaptation of the existing OSGi framework artifacts; these can then be used to expose an OSGi Framework manipulation API over JMX. See [JMX™ Management Model Specification](#) on page 389.

• **REST Management Service Specification** - REST is a powerful paradigm for accessing resources over a network and is widely used in Enterprise and Cloud settings as a protocol of choice, especially since it generally avoids problems with internet firewalls, from which other protocols may suffer. The REST Management Service Specification provides an API to manage and control an OSGi framework using REST operations. See [REST Management Service Specification](#) on page 657.

• **User Admin Service Specification** - The User Admin Service Specification provides authorization for OSGi framework actions based on authenticated users, instead of using the Java code-based permission model. See [User Admin Service Specification](#) on page 129.

• **Initial Provisioning Specification** - This specification defines how the Management Agent can make its way into the OSGi framework, and gives a structured view of the problems and their corresponding resolution methods. The purpose of this specification is to enable the management of a OSGi framework by an operator, and (optionally) to hand over the management of the OSGi framework later to another operator. See [Initial Provisioning Specification](#) on page 149.

• **Configuration Admin Service Specification** - The Configuration Admin service allows an operator to set the configuration information of bundles. See [Configuration Admin Service Specification](#) on page 63.

• **Metatype Service Specification** - The Metatype specification defines interfaces that allow bundle developers to describe attribute types in a computer readable form using metadata. It is mostly used in conjunction with the Configuration Admin Service. See [Metatype Service Specification](#) on page 107.

### 1.1.6 Naming and Directory services

Naming and directory services are well established and useful tools in enterprise applications. The Enterprise Specification includes the:

• **JNDI Services Specification** - The Java Naming and Directory Interface (JNDI) is a registry technology in Java applications, both in the Java SE and Java EE space. JNDI provides a vendor-neutral set of APIs that allow clients to interact with a naming service. See [JNDI Services Specification](#) on page 453.
1.1.7 Database Access

There are multiple approaches available to model and persist data in databases. The Enterprise Specification includes support for the common technologies:

- **JDBC™ Service Specification** - provides an API for applications to interact with relational database systems from different vendors. See *JDBC™ Service Specification* on page 445.
- **JPA Service Specification** - The Java Persistence API (JPA) is a specification that sets a standard for persistence in enterprise and non-enterprise JRE™-based environments. The JPA Service Specification defines how bundles may access and use JPA persistence units in applications, as well as how a JPA implementation can become available and be invoked within an OSGi framework. See *JPA Service Specification* on page 473.

1.1.8 Transaction Support

The support for transactions in Java is well defined outside of the OSGi specification. The Enterprise Specification includes the:

- **JTA Transaction Services Specification** - This specification provides the User Transaction, Transaction Manager, and Synchronization Registry services, which are based on their counterparts in the Java EE™ JTA Specifications. These services can be used to demarcate transaction boundaries, enlists durable and volatile resources, and provides transactional aware code to influence the outcome of a transaction and synchronize with the ending of a transaction. See *JTA Transaction Services Specification* on page 379.

1.1.9 Miscellaneous Supporting Services

Services providing solutions to common infrastructure requirements include:

- **Log Service Specification** - Provides a general purpose message logger for the OSGi framework. See *Log Service Specification* on page 35.
- **XML Parser Service Specification** - Addresses how the classes defined in JAXP can be used in an OSGi framework. See *XML Parser Service Specification* on page 689.
- **Service Loader Mediator Specification** - Addresses common problems of bundles that rely on the JRE provided Service Loader API to load custom Service Provider Implementations. This specification describes how to use the service registry for lookup of Service Providers as well as a solution for existing code to continue functioning using Service Loader API in a OSGi environment. See *Service Loader Mediator Specification* on page 551.
- **Coordinator Specification** - The Coordinator service provides a mechanism for multiple parties to collaborate on a common task without a priori knowledge of who will collaborate in that task. The service provides a rendezvous for an initiator to create a Coordination where collaborators can decide to participate. When the Coordination is ended, all participants are informed. See *Coordinator Service Specification* on page 503.

1.2 Application and Provisioning Support

The support for applications in OSGi environments has long been missing. The term ‘application’ may mean different things to different people, therefore, rather than defining what an application is, the OSGI specification provides a set of enabling services and specifications to aid in the definition, composition, deployment, and governance of a group of bundles and resources in an OSGi environment. The specifications are essential building blocks and provide from low level resolution to higher level composition abstractions that a management agent can use to build the necessary tools for managing OSGi applications.
• **Resolver Hook Service Specification** - An enabling technology defined by OSGi Core Release 6 for OSGi frameworks to support isolation of bundles.

• **Resolver Specification** - A service that can be the base for provisioning, deployment, build, and diagnostic tooling. Based on the generic Requirement/Capability model from OSGi Core Release 6 a management agent can use the Resolver service to compute the set of necessary resources needed to satisfy the given set of requirements. These requirements could represent the dependencies of a core set of bundles that need to be deployed. While the framework already hosts a Resolver for Bundle wiring, this specification makes a generic Resolver available as a service to resolve dependencies on resources beyond Bundle wiring. See **Resolver Service Specification** on page 639.

• **Repository Specification** - The Repository specification provides a standard API to access (possibly remote) repositories. Resources can be obtained from the repository by specifying declarative requirements, which might include for example 'provide all resources that export a given package' or 'provide the bundle with the given symbolic name and version', but can also be used with any other generic capabilities. While the Repository API can be used on its own, in conjunction with the Resolver Specification it provides the capability to manage retrieval of external resources during the resolution process. Typically one or multiple repositories provide the metadata for the Resolver service to draw the resolution from. A management agent can then use the repositories to apply the result of the resolution. See **Repository Service Specification** on page 531.

• **Subsystems Specification** - The core framework defines the life cycle of bundles and their relationships, but it is missing the ability to define a common life cycle and scoping rules for a set of bundles that are conceptually tied together. The Subsystems Specification provides the ability not only to group multiple bundles into a single manageable entity, but furthermore include arbitrary resources in this grouping. This allows for complete isolation as well as various sharing models of code, services, and resources through a management agent. See **Subsystem Service Specification** on page 565.

### 1.3 Reader Level

This specification is written for the following audiences:

- Application developers
- Framework and system service developers (system developers)
- Architects

This specification assumes that the reader has at least one year of practical experience in writing Java programs. Experience with enterprise systems and server environments is a plus. Application developers must be aware that the OSGi environment is significantly more dynamic than traditional desktop or server environments.

System developers require a very deep understanding of Java. At least three years of Java coding experience in a system environment is recommended. A Framework implementation will use areas of Java that are not normally encountered in traditional applications. Detailed understanding is required of class loaders, garbage collection, Java 2 security, and Java native library loading.

Architects should focus on the introduction of each subject. This introduction contains a general overview of the subject, the requirements that influenced its design, and a short description of its operation as well as the entities that are used. The introductory sections require knowledge of Java concepts like classes and interfaces, but should not require coding experience.

Most of these specifications are equally applicable to application developers and system developers.
1.4 Version Information

This document is the Enterprise Specification for the OSGi Enterprise Release 6.

1.4.1 OSGi Core Release 6

This specification is based on the OSGi Core Release 6. This specification can be downloaded from:

http://www.osgi.org/Specifications/HomePage

1.4.2 Component Versions

Components in this specification have their own specification version, independent of this specification. The following table summarizes the packages and specification versions for the different subjects.

<table>
<thead>
<tr>
<th>Item</th>
<th>Package(s)</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>-</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>101</td>
<td>org.osgi.service.log</td>
<td>Version 1.3</td>
</tr>
<tr>
<td>102</td>
<td>org.osgi.service.http</td>
<td>Version 1.2</td>
</tr>
<tr>
<td>104</td>
<td>org.osgi.service.cm</td>
<td>Version 1.5</td>
</tr>
<tr>
<td>105</td>
<td>org.osgi.service.metatype</td>
<td>Version 1.2</td>
</tr>
<tr>
<td>107</td>
<td>org.osgi.service.useradmin</td>
<td>Version 1.1</td>
</tr>
<tr>
<td>110</td>
<td>org.osgi.service.provisioning</td>
<td>Version 1.2</td>
</tr>
<tr>
<td>112</td>
<td>org.osgi.service.component</td>
<td>Version 1.2</td>
</tr>
<tr>
<td></td>
<td>org.osgi.service.component.annotations</td>
<td>Version 1.2</td>
</tr>
<tr>
<td>113</td>
<td>org.osgi.service.event</td>
<td>Version 1.3</td>
</tr>
<tr>
<td>121</td>
<td>org.osgi.service.blueprint.container</td>
<td>Version 1.0</td>
</tr>
<tr>
<td></td>
<td>org.osgi.service.blueprint.reflect</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>122</td>
<td>org.osgi.service.remoteserviceadmin</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>123</td>
<td>-</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>124</td>
<td>org.osgi.jmx</td>
<td>Version 1.1</td>
</tr>
<tr>
<td>125</td>
<td>org.osgi.service.jdbc</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>126</td>
<td>org.osgi.service.jndi</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>127</td>
<td>org.osgi.service.jpa</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>128</td>
<td>-</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>130</td>
<td>org.osgi.service.coordinator</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>132</td>
<td>org.osgi.service.repository</td>
<td>Version 1.1</td>
</tr>
<tr>
<td>133</td>
<td>org.osgi.service.serviceloader</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>134</td>
<td>org.osgi.service.subsystem</td>
<td>Version 1.1</td>
</tr>
<tr>
<td>135</td>
<td>org.osgi.namespace.contract</td>
<td>Version 1.0</td>
</tr>
<tr>
<td></td>
<td>org.osgi.namespace.extender</td>
<td>Version 1.0</td>
</tr>
<tr>
<td></td>
<td>org.osgi.namespace.service</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>136</td>
<td>org.osgi.service.resolver</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>137</td>
<td>org.osgi.service.rest</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>138</td>
<td>org.osgi.service.async</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>702</td>
<td>org.osgi.util.xml</td>
<td>Version 1.0</td>
</tr>
</tbody>
</table>
When a component is represented in a bundle, a version attribute is needed in the declaration of the Import-Package or Export-Package manifest headers.

1.4.3 Note

1. The org.osgi.jmx sub-packages are individually versioned to be aligned with the service they manage.

1.5 References

[1] Whiteboard Pattern

1.6 Changes

- Added REST Management Service Specification on page 657.
- Added Asynchronous Service Specification on page 677.
- Updated Subsystem Service Specification on page 565.
- Updated Repository Service Specification on page 531.
Remote Services

Version 1.0

The OSGi framework provides a local service registry for bundles to communicate through service objects, where a service is an object that one bundle registers and another bundle gets. A distribution provider can use this loose coupling between bundles to export a registered service by creating an endpoint. Vice versa, the distribution provider can create a proxy that accesses an endpoint and then registers this proxy as an imported service. A Framework can contain multiple distribution providers simultaneously, each independently importing and exporting services.

An endpoint is a communications access mechanisms to a service in another framework, a (web) service, another process, or a queue or topic destination, etc., requiring some protocol for communications. The constellation of the mapping between services and endpoints as well as their communication characteristics is called the topology. A common case for distribution providers is to be present on multiple frameworks importing and exporting services; effectively distributing the service registry.

The local architecture for remote services is depicted in Figure 100.1 on page 23.

100.1 The Fallacies

General abstractions for distributed systems have been tried before and often failed. Well known are the fallacies described in [1] The Fallacies of Distributed Computing Explained:

- The network is reliable
Remote Service Properties

- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There is one administrator
- Transport cost is zero
- The network is homogeneous

Most fallacies represent non-functional trade-offs that should be considered by administrators, their decisions can then be reflected in the topology. For example, in certain cases limited bandwidth is acceptable and the latency in a datacenter is near zero. However, the reliability fallacy is the hardest because it intrudes into the application code. If a communication channel is lost, the application code needs to take specific actions to recover from this failure.

This reliability aspect is also addressed with OSGi services because services are dynamic. Failures in the communications layer can be mapped to the unregistration of the imported service. OSGi bundles are already well aware of these dynamics, and a number of programming models have been developed to minimize the complexity of writing these dynamic applications.

100.2 Remote Service Properties

This section introduces a number of properties that participating bundles can use to convey information to the distribution provider according to this Remote Service specification.

The following table lists the properties that must be listed by a distribution provider.

<table>
<thead>
<tr>
<th>Service Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote.configs.supported</td>
<td>String</td>
<td>Registered by the distribution provider on one of its services to indicate the supported configuration types. See Configuration Types on page 29 and Dependencies on page 32.</td>
</tr>
<tr>
<td>remote.intents.supported</td>
<td>String</td>
<td>Registered by the distribution provider on one of its services to indicate the vocabulary of implemented intents. See Dependencies on page 32.</td>
</tr>
<tr>
<td>service.imported</td>
<td>*</td>
<td>Must be set by a distribution provider to any value when it registers the endpoint proxy as an imported service. A bundle can use this property to filter out imported services. The configuration information used to import this service, as described in service.exported.configs. Any associated properties for this configuration types must be properly mapped to the importing system. For example, a URL in these properties must point to a valid resource when used in the importing framework.</td>
</tr>
<tr>
<td>service.imported.configs</td>
<td>String</td>
<td>If multiple configuration types are listed in this property, then they must be synonyms for exactly the same remote endpoint that is used to export this service.</td>
</tr>
</tbody>
</table>
Service Property Name | Type   | Description
--- | --- | ---
service.intents   | String+ | A distribution provider must use this property to convey the combined intents of:

- The exporting service, and
- The intents that the exporting distribution provider adds.
- The intents that the importing distribution provider adds.

The properties for bundles providing services to be exported or require services to be imported are listed alphabetically in the following table. The scenarios that these properties are used in are discussed in later sections.

**Table 100.2 Remote Service Properties registered by Exporting bundles**

<table>
<thead>
<tr>
<th>Service Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service.exported.configs</td>
<td>String+</td>
<td>A list of configuration types that should be used to export the service. Each configuration type represents the configuration parameters for one or more Endpoints. A distribution provider should create endpoints for each configuration type that it supports. See Configuration Types on page 29 for more details. If this property is not set or empty a distribution provider is free to choose a default configuration type for the service.</td>
</tr>
<tr>
<td>service.exported.intents</td>
<td>String+</td>
<td>A list of intents that the distribution provider must implement to distribute the service. Intents listed in this property are reserved for intents that are critical for the code to function correctly, for example, ordering of messages. These intents should not be configurable. For more information about intents, see Intents on page 28. This property is optional.</td>
</tr>
<tr>
<td>service.exported.intents.extra</td>
<td>String+</td>
<td>This property is merged with the service.exported.intents property before the distribution provider interprets the listed intents; it has therefore the same semantics but the property should be configurable so the administrator can choose the intents based on the topology. Bundles should therefore make this property configurable, for example through the Configuration Admin service. See Intents on page 28. This property is optional. If absent or empty no specific intents are required.</td>
</tr>
<tr>
<td>service.exported.interfaces</td>
<td>String+</td>
<td>Setting this property marks this service for export. It defines the interfaces under which this service can be exported. This list must be a subset of the types listed in the objectClass service property. The single value of an asterisk (‘*’ Uno2A) indicates all interfaces in the registration's objectClass property and ignore the classes. It is strongly recommended to only export interfaces and not concrete classes due to the complexity of creating proxies for some type of concrete classes. See Registering a Service for Export on page 26.</td>
</tr>
</tbody>
</table>
Remote Service Properties

<table>
<thead>
<tr>
<th>Service Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service.intents</td>
<td>String+</td>
<td>A list of intents that this service implements. A distribution provider must use this property to convey the combined intents of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The exporting service, and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The intents that the exporting distribution provider adds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The intents that the importing distribution provider adds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To export a service, a distribution provider must expand any qualified intents to include those supported by the endpoint. This can be a subset of all known qualified intents. See Intents on page 28. This property is optional for registering bundles.</td>
</tr>
</tbody>
</table>

| service.pid           | String+ | Services that are exported should have a service.pid property. The service.pid (PID) is a unique persistent identity for the service, the PID is defined in Persistent Identifier (PID) of OSGi Core Release 6. This property enables a distribution provider to associate persistent proprietary data with a service registration. |

The properties and their treatment by the distribution provider is depicted in Figure 100.2.

Figure 100.2 Distribution Service Properties

100.2.1 Registering a Service for Export

A distribution provider should create one or more endpoints for an exported service when the following conditions are met:

• The service has the service property service.exported.interfaces set.
• All intents listed in service.exported.intents, service.exported.intents.extra and service.intents are part of the distributed provider's vocabulary
• None of the intents are mutually exclusive.
• The distribution provider can use the configuration types in service.exported.configs to create one or more endpoints.
The endpoint must at least implement all the intents that are listed in the `service.exported.intents` and `service.exported.intents.extra` properties.

The configuration types listed in the `service.exported.configs` can contain alternatives and/or synonyms. Alternatives describe different endpoints for the same service while a synonym describes a different configuration type for the same endpoint.

A distribution provider should create endpoints for each of the configuration types it supports; these configuration types should be alternatives. Synonyms are allowed.

If no configuration types are recognized, the distribution provider should create an endpoint with a default configuration type except when one of the listed configuration types is `<nodefault>`.

For more information about the configuration types, see further Configuration Types on page 29.

### 100.2.2 Getting an Imported Service

An imported service must be a normal service, there are therefore no special rules for getting it. An imported service has a number of additional properties that must be set by the distribution provider.

If the endpoint for an exported service is imported as an OSGi service in another framework, then the following properties must be treated as special.

- `service.imported` - Must be set to some value.
- `service.intents` - This must be the combination of the following:
  - The `service.intents` property on the exported service
  - The `service.exported.intents` and `service.exported.intents.extra` properties on the exported service
  - Any additional intents implemented by the distribution providers on both sides.
- `service.imported.configs` - Contains the configuration types that can be used to import this service. The types listed in this property must be synonymous, that is, they must refer to exactly the same endpoint that is exporting the service. See Configuration Types on page 29.
- `service.exported.*` - Properties starting with `service.exported` must not be set on the imported service.
- `service.exported.interfaces` - This property must not be set, its content is reflected in the `objectClass` property.

All other public service properties (not starting with a full stop (`
.\`)) must be listed on the imported service if they use the basic service property types. If the service property cannot be communicated because, for example, it uses a type that can not be marshalled by the distribution provider then the distribution provider must ignore this property.

The `service.imported` property indicates that a service is an imported service. If this service property is set to any value, then the imported service is a proxy for an endpoint. If a bundle wants to filter out imported services, then it can add the following filter:

```
(&(!{service.imported=*)) <previousfilter>)
```

Distribution providers can also use the Service Hook Service Specification of OSGi Core Release 6 to hide services from specific bundles.

### 100.2.3 On Demand Import

The Service Hooks Service Specification of OSGi Core Release 6, allows a distribution provider to detect when a bundle is listening for specific services. Bundles can request imported services with specific intents by building an appropriate filter. The distribution provider can use this information to import a service on demand.
The following example creates a Service Tracker that is interested in an imported service.

```java
Filter f = context.createFilter(
    "(& (objectClass=com.acme.Foo) "
    + " (service.intents=confidentiality)")
);
ServiceTracker tracker =
    new ServiceTracker(context, f, null);
tracker.open();
```

Such a Service Tracker will inform the Listener Hook and will give it the filter expression. If the distribution provider has registered such a hook, it will be informed about the need for an imported com.acme.Foo service that has a confidentiality intent. It can then use some proprietary means to find a service to import that matches the given object class and intent.

How the distribution provider finds an appropriate endpoint is out of scope for this specification.

### 100.3 Intents

An intent is a name for an abstract distribution capability. An intent can be *implemented* by a service; this can then be reflected in the service.intents property. An intent can also *constrain* the possible communication mechanisms that a distribution provider can choose to distribute a service. This is reflected in the service.export.intents and service.exported.intents.extra properties.

The purpose of the intents is to have a *vocabulary* that is shared between distribution aware bundles and the distribution provider. This vocabulary allows the bundles to express constraints on the export of their services as well as providing information on what intents are implemented by a service.

Intents have the following syntax:

```plaintext
intent  ::= token ( '.' token )?
```

*Qualified intents* use a full stop (`'.'`) to separate the intent from the qualifier. A qualifier provides additional details, however, it implies its prefix. For example:

```plaintext
confidentiality.message
```

This example can be *expanded* into confidentiality and confidentiality.message. Qualified intents can be used to provide additional details how an intent is achieved. However, a Distribution Provider must expand any qualified intents to include those supported by the endpoint. This can be a subset of all known qualified intents.

The concept of intents is derived from the [3] *SCA Policy Framework specification*. When designing a vocabulary for a distribution provider it is recommended to closely follow the vocabulary of intents defined in the SCA Policy Framework.

### 100.4 General Usage

#### 100.4.1 Call by Value

Normal service semantics are call-by-reference. An object passed as an argument in a service call is a direct reference to that object. Any changes to this object will be shared on both sides of the service registry.

Distributed services are different. Arguments are normally passed by value, which means that a copy is sent to the remote system, changes to this value are not reflected in the originating frame-
work. When using distributed services, call-by-value should always be assumed by all participants in the distribution chain.

100.4.2 Data Fencing

Services are syntactically defined by their Java interfaces. When exposing a service over a remote protocol, typically such an interface is mapped to a protocol-specific interface definition. For example, in CORBA the Java interfaces would be converted to a corresponding IDL definition. This mapping does not always result in a complete solution.

Therefore, for many practical distributed applications it will be necessary to constrain the possible usage of data types in service interfaces. A distribution provider must at least support interfaces (not classes) that only use the basic types as defined for the service properties. These are the primitive types and their wrappers as well as arrays and collections. See Filter Syntax of OSGi Core Release 6 for a list of service property types.

Distribution providers will in general provide a richer set of types that can be distributed.

100.4.3 Remote Services Life Cycle

If a distribution provider has distributed a service, it must closely track any modifications on the exported service. If there is a corresponding imported service, it must closely match any modified service properties in the way that was specified for the registration. If the exported service is unregistered, the endpoint must be withdrawn as soon as possible. If there is a corresponding imported service, then this imported service must also be unregistered expediently.

100.4.4 Runtime

An imported service is just like any other service and can be used as such. However, certain non-functional characteristics of this service can differ significantly from what is normal for an in-VM object call. Many of these characteristics can be mapped to the normal service operations. That is, if the connection fails in any way, the service can be unregistered. According to the standard OSGi contract, this means that the users of that service must perform the appropriate cleanup to prevent stale references.

100.4.5 Exceptions

It is impossible to guarantee that a service is not used when it is no longer valid. Even with the synchronous callbacks from the Service Listeners, there is always a finite window where a service can be used while the underlying implementation has failed. In a distributed environment, this window can actually be quite large for an imported service.

Such failure situations must be exposed to the application code that uses a failing imported service. In these occasions, the distribution provider must notify the application by throwing a Service Exception, or subclass thereof, with the reason REMOTE. The Service Exception is a Runtime Exception, it can be handled higher up in the call chain. The cause of this Service Exception must be the Exception that caused the problem.

A distribution provider should log any problems with the communications layer to the Log Service, if available.

100.5 Configuration Types

An exported service can have a service.exported.configs service property. This property lists configuration types for endpoints that are provided for this service. Each type provides a specification that defines how the configuration data for one or more endpoints is provided. For example, a hypothetical configuration type could use a service property to hold a URL for the RMI naming registry.
Configuration types that are not defined by the OSGi Alliance should use a name that follows the reverse capabilities domain name scheme defined in [4] *Java Language Specification* for Java packages. For example, `com.acme.wdsl` would be the proprietary way for the ACME company to specify a WSDL configuration type.

### 100.5.1 Configuration Type Properties

The `service.exported.configs` and `service.imported.configs` use the configuration types in very different ways. That is, the `service.imported.configs` property is not a copy of the `service.exported.configs` as the name might seem to imply.

An exporting service can list its desired configuration types in the `service.exported.configs` property. This property is potentially seen and interpreted by multiple distribution providers. Each of these providers can independently create endpoints from the configuration types. In principle, the `service.exported.configs` lists alternatives for a single distribution provider and can list synonyms to support alternative distribution providers. If only one of the synonyms is useful, there is an implicit assumption that when the service is exported, only one of the synonyms should be supported by the installed distribution providers. If it is detected that this assumption is violated, then an error should be logged and the conflicting configuration is further ignored.

The interplay of synonyms and alternatives is depicted in Table 100.3. In this table, the first columns on the left list different combinations of the configuration types in the `service.exported.configs` property. The next two columns list two distribution providers that each support an overlapping set of configuration types. The `x`'s in this table indicate if a configuration type or distribution provider is active in a line. The description then outlines the issues, if any. It is assumed in this table that hypothetical configuration types `net.rmi` and `com.rmix` map to an identical endpoint, just like `net.soap` and `net.soapx`.

**Table 100.3 Synonyms and Alternatives in Exported Configurations**

<table>
<thead>
<tr>
<th>service.exported.configs</th>
<th>Distribution Provider A</th>
<th>Distribution Provider B</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.rmi</td>
<td>Supports:</td>
<td>Supports:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>net.rmi</td>
<td>net.rmi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>com.rmix</td>
<td>com.soapx</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Ok, A will create an endpoint for the RMI and SOAP alternatives.</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Configuration error. There is a clash for <code>net.rmi</code> because A and B can both create an endpoint for the same configuration. It is likely that one will fail.</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Ok, exported on com.soapx by A, the <code>net.soap</code> is ignored.</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Synonym error because A and B export to same SOAP endpoint, it is likely that one will fail.</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Ok, two alternative endpoints over RMI (by A) and SOAP (by B) are created. This is a typical use case.</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Ok: Synonyms are used to allow frameworks that have either A or B installed. In this case A exports over SOAP.</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Ok: Synonyms are used to allow frameworks that have either A or B installed. In this case B exports.</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Ok: A creates an endpoint with default configuration type.</td>
</tr>
</tbody>
</table>
To summarize, the following rules apply for a single distribution provider:

- Only configuration types that are supported by this distribution provider must be used. All other configuration types must be ignored.
- All of the supported configuration types must be alternatives, that is, they must map to different endpoints. Synonyms for the same distribution provider should be logged as errors.
- If a configuration type results in an endpoint that is already in use, then an error should be logged. It is likely then that another distribution provider already had created that endpoint.

An export of a service can therefore result in multiple endpoints being created. For example, a service can be exported over RMI as well as SOAP. Creating an endpoint can fail, in that case the distribution provider must log this information in the Log Service, if available, and not export the service to that endpoint. Such a failure can, for example, occur when two configuration types are synonym and multiple distribution providers are installed that supporting this type.

On the importing side, the `service.imported.configs` property lists configuration types that must refer to the same endpoint. That is, it can list alternative configuration types for this endpoint but all configuration types must result in the same endpoint.

For example, there are two distribution providers installed at the exporting and importing frameworks. Distribution provider A supports the hypothetical configuration type `net.rmi` and `net.soap`. Distribution provider B supports the hypothetical configuration type `net.smart`. A service is registered that lists all three of those configuration types.

Distribution provider A will create two endpoints, one for RMI and one for SOAP. Distribution provider B will create one endpoint for the smart protocol. The distribution provider A knows how to create the configuration data for the `com.acme.rmi` configuration type as well and can therefore create a synonymous description of the endpoint in that configuration type. It will therefore set the imported configuration type for the RMI endpoint to:

```
service.imported.configs = net.rmi, com.acme.rmi
net.rmi.url = rmi://172.25.25.109:1099/service-id/24
com.acme.rmi.address = 172.25.25.109
com.acme.rmi.port = 1099
com.acme.rmi.path = service-id/24
```
100.5.2 Dependencies

A bundle that uses a configuration type has an implicit dependency on the distribution provider. To make this dependency explicit, the distribution provider must register a service with the following properties:

- `remote.intents.supported` (String+) The vocabulary of the given distribution provider.
- `remote.configs.supported` (String+) The configuration types that are implemented by the distribution provider.

A bundle that depends on the availability of specific intents or configuration types can create a service dependency on an anonymous service with the given properties. The following filter is an example of depending on a hypothetical `net.rmi` configuration type:

```
(remote.configs.supported=net.rmi)
```

100.6 Security

The distribution provider will be required to invoke methods on any exported service. This implies that it must have the combined set of permissions of all methods it can call. It also implies that the distribution provider is responsible for ensuring that a bundle that calls an imported service is not granted additional permissions through the fact that the distribution provider will call the exported service, not the original invoker.

The actual mechanism to ensure that bundles can get additional permissions through the distribution is out of scope for this specification. However, distribution providers should provide mechanisms to limit the set of available permissions for a remote invocation, preferably on a small granularity basis.

One possible means is to use the `getAccessControlContext` method on the Conditional Permission Admin service to get an Access Control Context that is used in a `doPrivileged` block where the invocation takes place. The `getAccessControlContext` method takes a list of signers which could repre-
sent the remote bundles that cause an invocation. How these are authenticated is up to the distribution provider.

A distribution provider is a potential attack point for intruders. Great care should be taken to properly setup the permissions or topology in an environment that requires security.

100.6.1 Limiting Exports and Imports

Service registration and getting services is controlled through the ServicePermission class. This permission supports a filter based constructor that can assert service properties. This facility can be used to limit bundles from being able to register exported services or get imported services if they are combined with Conditional Permission Admin's ALLOW facility. The following example shows how all bundles except from www.acme.com are denied the registration and getting of distributed services.

```java
DENY {
  [...BundleLocationCondition("http://www.acme.com/*" "!")]
  (...ServicePermission "(service.imported=*)" "GET" )
  (...ServicePermission "(service.exported.interfaces=*)"
      "REGISTER" )
}
```

100.7 References

http://www.rgoarchitects.com/Files/fallacies.pdf

http://www.oasis-opencsa.org/

http://www.oasis-open.org/committees/sca-policy/

http://docs.oracle.com/javase/specs/
101 Log Service Specification

Version 1.3

101.1 Introduction

The Log Service provides a general purpose message logger for the OSGi framework. It consists of two services, one for logging information and another for retrieving current or previously recorded log information.

This specification defines the methods and semantics of interfaces which bundle developers can use to log entries and to retrieve log entries.

Bundles can use the Log Service to log information for the Operator. Other bundles, oriented toward management of the environment, can use the Log Reader Service to retrieve Log Entry objects that were recorded recently or to receive Log Entry objects as they are logged by other bundles.

101.1.1 Entities

- **LogService** - The service interface that allows a bundle to log information, including a message, a level, an exception, a ServiceReference object, and a Bundle object.
- **LogEntry** - An interface that allows access to a log entry in the log. It includes all the information that can be logged through the Log Service and a time stamp.
- **LogReaderService** - A service interface that allows access to a list of recent LogEntry objects, and allows the registration of a LogListener object that receives LogEntry objects as they are created.
- **LogListener** - The interface for the listener to LogEntry objects. Must be registered with the Log Reader Service.

![Log Service Class Diagram](image)

*Figure 101.1 Log Service Class Diagram org.osgi.service.log package*
101.2 The Log Service Interface

The LogService interface allows bundle developers to log messages that can be distributed to other bundles, which in turn can forward the logged entries to a file system, remote system, or some other destination.

The LogService interface allows the bundle developer to:

- Specify a message and/or exception to be logged.
- Supply a log level representing the severity of the message being logged. This should be one of the levels defined in the LogService interface but it may be any integer that is interpreted in a user-defined way.
- Specify the Service associated with the log requests.

By obtaining a LogService object from the Framework service registry, a bundle can start logging messages to the LogService object by calling one of the LogService methods. A Log Service object can log any message, but it is primarily intended for reporting events and error conditions.

The LogService interface defines these methods for logging messages:

- `log(int, String)` - This method logs a simple message at a given log level.
- `log(int, String, Throwable)` - This method logs a message with an exception at a given log level.
- `log(ServiceReference, int, String)` - This method logs a message associated with a specific service.
- `log(ServiceReference, int, String, Throwable)` - This method logs a message with an exception associated with a specific service.

While it is possible for a bundle to call one of the log methods without providing a ServiceReference object, it is recommended that the caller supply the ServiceReference argument whenever appropriate, because it provides important context information to the operator in the event of problems.

The following example demonstrates the use of a log method to write a message into the log:

```java
logService.log(
    myServiceReference,
    LogService.LOG_INFO,
    "myService is up and running"
);
```

In the example, the myServiceReference parameter identifies the service associated with the log request. The specified level, LogService.LOG_INFO, indicates that this message is informational.

The following example code records error conditions as log messages.

```java
try {
    FileInputStream fis = new FileInputStream("myFile");
    int b;
    while ( (b = fis.read()) != -1 ) {
        ...
    }
    fis.close();
} catch ( IOException exception ) {
    logService.log(
        myServiceReference,
```
Notice that in addition to the error message, the exception itself is also logged. Providing this information can significantly simplify problem determination by the Operator.

### 101.3 Log Level and Error Severity

The log methods expect a log level indicating error severity, which can be used to filter log messages when they are retrieved. The severity levels are defined in the LogService interface.

Callers must supply the log levels that they deem appropriate when making log requests.

The following table lists the log levels.

<table>
<thead>
<tr>
<th>Level</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_DEBUG</td>
<td>Used for problem determination and may be irrelevant to anyone but the bundle developer.</td>
</tr>
<tr>
<td>LOG_ERROR</td>
<td>Indicates the bundle or service may not be functional. Action should be taken to correct this situation.</td>
</tr>
<tr>
<td>LOG_INFO</td>
<td>May be the result of any change in the bundle or service and does not indicate a problem.</td>
</tr>
<tr>
<td>LOG_WARNING</td>
<td>Indicates a bundle or service is still functioning but may experience problems in the future because of the warning condition.</td>
</tr>
</tbody>
</table>

### 101.4 Log Reader Service

The Log Reader Service maintains a list of LogEntry objects called the log. The Log Reader Service is a service that bundle developers can use to retrieve information contained in this log, and receive notifications about LogEntry objects when they are created through the Log Service.

The size of the log is implementation-specific, and it determines how far into the past the log entries go. Additionally, some log entries may not be recorded in the log in order to save space. In particular, LOG_DEBUG log entries may not be recorded. Note that this rule is implementation-dependent. Some implementations may allow a configurable policy to ignore certain LogEntry object types.

The LogReaderService interface defines these methods for retrieving log entries.

- `getLog()` - This method retrieves past log entries as an enumeration with the most recent entry first.
- `addLogListener(LogListener)` - This method is used to subscribe to the Log Reader Service in order to receive log messages as they occur. Unlike the previously recorded log entries, all log messages must be sent to subscribers of the Log Reader Service as they are recorded.

A subscriber to the Log Reader Service must implement the LogListener interface.

After a subscription to the Log Reader Service has been started, the subscriber's LogListener.logged method must be called with a Log-Entry object for the message each time a message is logged.

The LogListener interface defines the following method:


- `logged(LogEntry)` - This method is called for each `Log-Entry` object created. A Log Reader Service implementation must not filter entries to the `LogListener` interface as it is allowed to do for its log. A `LogListener` object should see all `LogEntry` objects that are created.

The delivery of `LogEntry` objects to the `LogListener` object should be done asynchronously.

### 101.5 Log Entry Interface

The `LogEntry` interface abstracts a log entry. It is a record of the information that was passed when an event was logged, and consists of a superset of information which can be passed through the `LogService` methods. The `LogEntry` interface defines these methods to retrieve information related to `Log-Entry` objects:

- `getBundle()` - This method returns the Bundle object related to a `Log-Entry` object.
- `getException()` - This method returns the exception related to a `Log-Entry` object. In some implementations, the returned exception may not be the original exception. To avoid references to a bundle defined exception class, thus preventing an uninstalled bundle from being garbage collected, the Log Service may return an exception object of an implementation defined ` Throwable` subclass. This object will attempt to return as much information as possible, such as the message and stack trace, from the original exception object.
- `getLevel()` - This method returns the severity level related to a `Log-Entry` object.
- `getMessage()` - This method returns the message related to a `Log-Entry` object.
- `getServiceReference()` - This method returns the `ServiceReference` object of the service related to a `Log-Entry` object.
- `getTime()` - This method returns the time that the log entry was created.

### 101.6 Mapping of Events

Implementations of a Log Service must log Framework-generated events and map the information to `LogEntry` objects in a consistent way. Framework events must be treated exactly the same as other logged events and distributed to all `LogListener` objects that are associated with the Log Reader Service. The following sections define the mapping for the three different event types: Bundle, Service, and Framework.

#### 101.6.1 Bundle Events Mapping

A Bundle Event is mapped to a `LogEntry` object according to the following table.

<table>
<thead>
<tr>
<th><code>Log Entry method</code></th>
<th>Information about Bundle Event</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getLevel()</code></td>
<td>LOG_INFO</td>
</tr>
<tr>
<td><code>getBundle()</code></td>
<td>Identifies the bundle to which the event happened. In other words, it identifies the bundle that was installed, started, stopped, updated, or uninstalled. This identification is obtained by calling <code>getBundle()</code> on the <code>BundleEvent</code> object.</td>
</tr>
<tr>
<td><code>getException()</code></td>
<td>null</td>
</tr>
<tr>
<td><code>getServiceReference()</code></td>
<td>null</td>
</tr>
</tbody>
</table>
**Log Service Specification Version 1.3**

**Mapping of Events**

---

Log Entry method | Information about Bundle Event
--- | ---
getMessage() | The message depends on the event type:
  - INSTALLED - "BundleEvent INSTALLED"
  - STARTED - "BundleEvent STARTED"
  - STOPPED - "BundleEvent STOPPED"
  - UPDATED - "BundleEvent UPDATED"
  - UNINSTALLED - "BundleEvent UNINSTALLED"
  - RESOLVED - "BundleEvent RESOLVED"
  - UNRESOLVED - "BundleEvent UNRESOLVED"

101.6.2 **Service Events Mapping**

A Service Event is mapped to a LogEntry object according to the following table.

**Table 101.3** Mapping of Service Events to Log Entries

<table>
<thead>
<tr>
<th>Log Entry method</th>
<th>Information about Service Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>getLevel()</td>
<td>LOG_INFO, except for the ServiceEvent.MODIFIED event. This event can happen frequently and contains relatively little information. It must be logged with a level of LOG_DEBUG.</td>
</tr>
<tr>
<td>getBundle()</td>
<td>Identifies the bundle that registered the service associated with this event. It is obtained by calling getServiceReference().getBundle() on the ServiceEvent object.</td>
</tr>
<tr>
<td>getException()</td>
<td>null</td>
</tr>
<tr>
<td>getServiceReference()</td>
<td>Identifies a reference to the service associated with the event. It is obtained by calling getServiceReference() on the ServiceEvent object.</td>
</tr>
<tr>
<td>getMessage()</td>
<td>This message depends on the actual event type. The messages are mapped as follows:</td>
</tr>
</tbody>
</table>
  - REGISTERED - "ServiceEvent REGISTERED"
  - MODIFIED - "ServiceEvent MODIFIED"
  - UNREGISTERING - "ServiceEvent UNREGISTERING"

101.6.3 **Framework Events Mapping**

A Framework Event is mapped to a LogEntry object according to the following table.

**Table 101.4** Mapping of Framework Event to Log Entries

<table>
<thead>
<tr>
<th>Log Entry method</th>
<th>Information about Framework Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>getLevel()</td>
<td>LOG_INFO, except for the FrameworkEvent.ERROR event. This event represents an error and is logged with a level of LOG_ERROR.</td>
</tr>
<tr>
<td>getBundle()</td>
<td>Identifies the bundle associated with the event. This may be the system bundle. It is obtained by calling getBundle() on the FrameworkEvent object.</td>
</tr>
<tr>
<td>getException()</td>
<td>Identifies the exception associated with the error. This will be null for event types other than ERROR. It is obtained by calling getThrowable() on the FrameworkEvent object.</td>
</tr>
<tr>
<td>getServiceReference()</td>
<td>null</td>
</tr>
</tbody>
</table>
Log Entry method
getMessage()

Information about Framework Event
This message depends on the actual event type. The messages are mapped as follows:

- `STARTED` - "FrameworkEvent STARTED"
- `ERROR` - "FrameworkEvent ERROR"
- `PACKAGES_REFRESHED` - "FrameworkEvent PACKAGES REFRESHED"
- `STARTLEVEL_CHANGED` - "FrameworkEvent STARTLEVEL CHANGED"
- `WARNING` - "FrameworkEvent WARNING"
- `INFO` - "FrameworkEvent INFO"

101.6.4 Log Events
Log events must be delivered by the Log Service implementation to the Event Admin service (if present) asynchronously under the topic:

```
org/osgi/service/log/LogEntry/<event type>
```

The logging level is used as event type:

- `LOG_ERROR`
- `LOG_WARNING`
- `LOG_INFO`
- `LOG_DEBUG`
- `LOG_OTHER` (when event is not recognized)

The properties of a log event are:

- `bundle.id` - (Long) The source bundle's id.
- `bundle.symbolicName` - (String) The source bundle's symbolic name. Only set if not null.
- `bundle` - (Bundle) The source bundle.
- `log.level` - (Integer) The log level.
- `message` - (String) The log message.
- `timestamp` - (Long) The log entry's timestamp.
- `log.entry` - (LogEntry) The LogEntry object.

If the log entry has an associated Exception:

- `exception.class` - (String) The fully-qualified class name of the attached exception. Only set if the getException method returns a non-null value.
- `exception.message` - (String) The message of the attached Exception. Only set if the Exception message is not null.
- `exception` - (Throwable) The Exception returned by the getException method.

If the getServiceReference method returns a non-null value:

- `service` - (ServiceReference) The result of the getServiceReference method.
- `service.id` - (Long) The id of the service.
- `service.pid` - (String) The service's persistent identity. Only set if the service.pid service property is not null.
- `service.objectClass` - (String[]) The object class of the service object.
## 101.7 Security

The Log Service should only be implemented by trusted bundles. This bundle requires ServicePermission[LogService|LogReaderService, REGISTER]. Virtually all bundles should get ServicePermission[LogService, GET]. The ServicePermission[LogReaderService, GET] should only be assigned to trusted bundles.

## 101.8 org.osgi.service.log

Log Service Package Version 1.3.

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.log; version="[1.3,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.log; version="[1.3,1.4)"
```

### 101.8.1 Summary

- **LogEntry** - Provides methods to access the information contained in an individual Log Service log entry.
- **LogListener** - Subscribes to LogEntry objects from the LogReaderService.
- **LogReaderService** - Provides methods to retrieve LogEntry objects from the log.
- **LogService** - Provides methods for bundles to write messages to the log.

### 101.8.2 public interface LogEntry

Provides methods to access the information contained in an individual Log Service log entry.

A LogEntry object may be acquired from the LogReaderService.getLog method or by registering a LogListener object.

**See Also**

LogReaderService.getLog, LogListener

**Concurrency**

Thread-safe

**No Implement**

Consumers of this API must not implement this interface

#### 101.8.2.1 public Bundle getBundle()

- Returns the bundle that created this LogEntry object.

**Returns**

The bundle that created this LogEntry object; null if no bundle is associated with this LogEntry object.

#### 101.8.2.2 public Throwable getException()

- Returns the exception object associated with this LogEntry object.

In some implementations, the returned exception may not be the original exception. To avoid references to a bundle defined exception class, thus preventing an uninstalled bundle from being garbage collected, the Log Service may return an exception object of an implementation defined Throwable subclass. The returned object will attempt to provide as much information as possible from the original exception object such as the message and stack trace.
Returns Throwable object of the exception associated with this LogEntry; null if no exception is associated with this LogEntry object.

101.8.2.3 public int getLevel()

☐ Returns the severity level of this LogEntry object.
This is one of the severity levels defined by the LogService interface.

Returns Severity level of this LogEntry object.
See Also LogService.LOG_ERROR, LogService.LOG_WARNING, LogService.LOG_INFO, LogService.LOG_DEBUG

101.8.2.4 public String getMessage()

☐ Returns the human readable message associated with this LogEntry object.

Returns String containing the message associated with this LogEntry object.

101.8.2.5 public ServiceReference getServiceReference()

☐ Returns the ServiceReference object for the service associated with this LogEntry object.

Returns ServiceReference object for the service associated with this LogEntry object; null if no ServiceReference object was provided.

101.8.2.6 public long getTime()

☐ Returns the value of currentTimeMillis() at the time this LogEntry object was created.

Returns The system time in milliseconds when this LogEntry object was created.
See Also System.currentTimeMillis()

101.8.3 public interface LogListener
extends EventListener

Subscribes to LogEntry objects from the LogReaderService.
A LogListener object may be registered with the Log Reader Service using the
LogReaderService.addLogListener method. After the listener is registered, the logged method will be called for each LogEntry object created. The LogListener object may be unregistered by calling the
LogReaderService.removeLogListener method.
See Also LogReaderService, LogEntry, LogReaderService.addLogListener(LogListener),
LogReaderService.removeLogListener(LogListener)

Concurrency Thread safe

101.8.3.1 public void logged(LogEntry entry)

entry A LogEntry object containing log information.

☐ Listener method called for each LogEntry object created.
As with all event listeners, this method should return to its caller as soon as possible.
See Also LogEntry

101.8.4 public interface LogReaderService

Provides methods to retrieve LogEntry objects from the log.
There are two ways to retrieve LogEntry objects:

• The primary way to retrieve LogEntry objects is to register a LogListener object whose
LogListener.logged method will be called for each entry added to the log.
• To retrieve past LogEntry objects, the getLog method can be called which will return an Enumeration of all LogEntry objects in the log.

See Also  LogEntry, LogListener, LogListener.logged(LogEntry)

Concurrency  Thread-safe

101.8.4.1  public void addLogListener(LogListener listener)

listener  A LogListener object to register; the LogListener object is used to receive LogEntry objects.

□ Subscribes to LogEntry objects.

This method registers a LogListener object with the Log Reader Service. The LogListener.logged(LogEntry) method will be called for each LogEntry object placed into the log.

When a bundle which registers a LogListener object is stopped or otherwise releases the Log Reader Service, the Log Reader Service must remove all of the bundle's listeners.

If this Log Reader Service's list of listeners already contains a listener \( l \) such that \( (l==\text{listener}) \), this method does nothing.

See Also  LogListener, LogEntry, LogListener.logged(LogEntry)

101.8.4.2  public Enumeration getLog()

□ Returns an Enumeration of all LogEntry objects in the log.

Each element of the enumeration is a LogEntry object, ordered with the most recent entry first.

Whether the enumeration is of all LogEntry objects since the Log Service was started or some recent past is implementation-specific. Also implementation-specific is whether informational and debug LogEntry objects are included in the enumeration.

Returns  An Enumeration of all LogEntry objects in the log.

101.8.4.3  public void removeLogListener(LogListener listener)

listener  A LogListener object to unregister.

□ Unsubscribes to LogEntry objects.

This method unregisters a LogListener object from the Log Reader Service.

If listener is not contained in this Log Reader Service's list of listeners, this method does nothing.

See Also  LogListener

101.8.5  public interface LogService

Provides methods for bundles to write messages to the log.

LogService methods are provided to log messages; optionally with a ServiceReference object or an exception.

Bundles must log messages in the OSGi environment with a severity level according to the following hierarchy:

1. LOG_ERROR
2. LOG_WARNING
3. LOG_INFO
4. LOG_DEBUG

Concurrency  Thread-safe

No Implement  Consumers of this API must not implement this interface
101.8.5.1  public static final int LOG_DEBUG = 4
A debugging message (Value 4).
This log entry is used for problem determination and may be irrelevant to anyone but the bundle developer.

101.8.5.2  public static final int LOG_ERROR = 1
An error message (Value 1).
This log entry indicates the bundle or service may not be functional.

101.8.5.3  public static final int LOG_INFO = 3
An informational message (Value 3).
This log entry may be the result of any change in the bundle or service and does not indicate a problem.

101.8.5.4  public static final int LOG_WARNING = 2
A warning message (Value 2).
This log entry indicates a bundle or service is still functioning but may experience problems in the future because of the warning condition.

101.8.5.5  public void log(int level,String message)
level The severity of the message. This should be one of the defined log levels but may be any integer that is interpreted in a user defined way.
message Human readable string describing the condition or null.
   □ Logs a message.
   The ServiceReference field and the Throwable field of the LogEntry object will be set to null.
See Also LOG_ERROR, LOG_WARNING, LOG_INFO, LOG_DEBUG

101.8.5.6  public void log(int level,String message,Throwable exception)
level The severity of the message. This should be one of the defined log levels but may be any integer that is interpreted in a user defined way.
message The human readable string describing the condition or null.
exception The exception that reflects the condition or null.
   □ Logs a message with an exception.
   The ServiceReference field of the LogEntry object will be set to null.
See Also LOG_ERROR, LOG_WARNING, LOG_INFO, LOG_DEBUG

101.8.5.7  public void log(ServiceReference sr,int level,String message)
sr The ServiceReference object of the service that this message is associated with or null.
level The severity of the message. This should be one of the defined log levels but may be any integer that is interpreted in a user defined way.
message Human readable string describing the condition or null.
   □ Logs a message associated with a specific ServiceReference object.
   The Throwable field of the LogEntry will be set to null.
See Also LOG_ERROR, LOG_WARNING, LOG_INFO, LOG_DEBUG
101.8.5.8 public void log(ServiceReference sr, int level, String message, Throwable exception)

  sr  The ServiceReference object of the service that this message is associated with.
  
  level The severity of the message. This should be one of the defined log levels but may be any integer that
  is interpreted in a user defined way.
  
  message Human readable string describing the condition or null.
  
  exception The exception that reflects the condition or null.
  
  - Logs a message with an exception associated and a ServiceReference object.

See Also  LOG_ERROR, LOG_WARNING, LOG_INFO, LOG_DEBUG
Http Service Specification

Version 1.2

102.1 Introduction

An OSGi framework normally provides users with access to services on the Internet and other networks. This access allows users to remotely retrieve information from, and send control to, services in an OSGi framework using a standard web browser.

Bundle developers typically need to develop communication and user interface solutions for standard technologies such as HTTP, HTML, XML, and servlets.

The Http Service supports two standard techniques for this purpose:

- **Registering servlets** - A servlet is a Java object which implements the Java Servlet API. Registering a servlet in the Framework gives it control over some part of the Http Service URI name-space.
- **Registering resources** - Registering a resource allows HTML files, image files, and other static resources to be made visible in the Http Service URI name-space by the requesting bundle.

Implementations of the Http Service can be based on:

- [1] *HTTP 1.0 Specification RFC-1945*
- [2] *HTTP 1.1 Specification RFC-2616*

Alternatively, implementations of this service can support other protocols if these protocols can conform to the semantics of the javax.servlet API. This additional support is necessary because the Http Service is closely related to [3] *Java Servlet Technology*. Http Service implementations must support at least version 2.1 of the Java Servlet API.

102.1.1 Entities

This specification defines the following interfaces which a bundle developer can implement collectively as an Http Service or use individually:

- **HttpContext** - Allows bundles to provide information for a servlet or resource registration.
- **HttpService** - Allows other bundles in the Framework to dynamically register and unregister resources and servlets into the Http Service URI name-space.
- **NamespaceException** - Is thrown to indicate an error with the caller's request to register a servlet or resource into the Http Service URI name-space.
102.2 Registering Servlets

javax.servlet.Servlet objects can be registered with the Http Service by using the
HttpService interface. For this purpose, the HttpService interface defines the method
registerServlet(String, javax.servlet.Servlet, Dictionary, HttpContext).

For example, if the Http Service implementation is listening to port 80 on the machine
www.acme.com and the Servlet object is registered with the name "/servlet", then the Servlet
object's service method is called when the following URL is used from a web browser:

http://www.acme.com/servlet?name=bugs

All Servlet objects and resource registrations share the same name-space. If an attempt is made
to register a resource or Servlet object under the same name as a currently registered resource or
Servlet object, a NamespaceException is thrown. See Mapping HTTP Requests to Servlet and Resource
Registrations on page 51 for more information about the handling of the Http Service name-
space.

Each Servlet registration must be accompanied with an HttpContext object. This object provides
the handling of resources, media typing, and a method to handle authentication of remote requests.
See Authentication on page 54.

For convenience, a default HttpContext object is provided by the Http Service and can be obtained
with createDefaultHttpContext(). Passing a null parameter to the registration method achieves the
same effect.

Servlet objects require a ServletContext object. This object provides a number of functions to access
the Http Service Java Servlet environment. It is created by the implementation of the Http Service
for each unique HttpContext object with which a Servlet object is registered. Thus, Servlet objects
registered with the same HttpContext object must also share the same ServletContext object.
Servlet objects are initialized by the Http Service when they are registered and bound to that specific Http Service. The initialization is done by calling the Servlet object's Servlet.init(ServletConfig) method. The ServletConfig parameter provides access to the initialization parameters specified when the Servlet object was registered.

Therefore, the same Servlet instance must not be reused for registration with another Http Service, nor can it be registered under multiple names. Unique instances are required for each registration.

The following example code demonstrates the use of the registerServlet method:

```java
Hashtable initparams = new Hashtable();
initparams.put( "name", "value" );

Servlet myServlet = new HttpServlet() {
    String name = "<not set>";

    public void init( ServletConfig config ) {
        this.name = (String)
            config.getInitParameter( "name" );
    }

    public void doGet(
        HttpServletRequest req,
        HttpServletResponse rsp
    ) throws IOException {
        rsp.setContentType( "text/plain" );
        req.getWriter().println( this.name );
    }
};

getHttpService().registerServlet(
    new HttpServlet() {
        String name = "<not set>";
    },
    "/servletAlias",
    myServlet,
    initparams,
    null // use default context
);
// myServlet has been registered
// and its init method has been called. Remote
// requests are now handled and forwarded to
// the servlet.
...
getHttpService().unregister("/servletAlias");
// myServlet has been unregistered and its
// destroy method has been called
```

This example registers the servlet, myServlet, at alias:/servletAlias. Future requests for http://www.acme.com/servletAlias maps to the servlet, myServlet, whose service method is called to process the request. (The service method is called in the HttpServlet base class and dispatched to a doGet, doPost, doPost, doOptions, doTrace, or doDelete call depending on the HTTP request method used.)
102.3 Registering Resources

A resource is a file containing images, static HTML pages, sounds, movies, applets, etc. Resources do not require any handling from the bundle. They are transferred directly from their source—usually the JAR file that contains the code for the bundle—to the requestor using HTTP.

Resources could be handled by Servlet objects as explained in Registering Servlets on page 48. Transferring a resource over HTTP, however, would require very similar Servlet objects for each bundle. To prevent this redundancy, resources can be registered directly with the Http Service via the HttpService interface. This HttpService interface defines the registerResources(String,String,HttpContext) method for registering a resource into the Http Service URI name-space.

The first parameter is the external alias under which the resource is registered with the Http Service. The second parameter is an internal prefix to map this resource to the bundle’s name-space. When a request is received, the HttpService object must remove the external alias from the URI, replace it with the internal prefix, and call the getResource(String) method with this new name on the associated HttpContext object. The HttpContext object is further used to get the MIME type of the resource and to authenticate the request.

Resources are returned as a java.net.URL object. The Http Service must read from this URL object and transfer the content to the initiator of the HTTP request.

This return type was chosen because it matches the return type of the java.lang.Class.getResource(String resource) method. This method can retrieve resources directly from the same place as the one from which the class was loaded—often a package directory in the JAR file of the bundle. This method makes it very convenient to retrieve resources from the bundle that are contained in the package.

The following example code demonstrates the use of the register Resources method:

```java
package com.acme;
...
HttpContext context = new HttpContext() {
    public boolean handleSecurity(
            HttpServletRequest request,
            HttpServletResponse response
    ) throws IOException {
        return true;
    }

    public URL getResource(String name) {
        return getClass().getResource(name);
    }

    public String getMimeType(String name) {
        return null;
    }
};

getHttpService().registerResources (  
    "/files",
    "www",
    context
);  
...
getHttpService().unregister("/files");

This example registers the alias /files on the Http Service. Requests for resources below this namespace are transferred to the HttpContext object with an internal name of www/<name>. This example uses the Class.getResource(String) method. Because the internal name does not start with a "/", it must map to a resource in the "com/acme/www" directory of the JAR file. If the internal name did start with a "/", the package name would not have to be prefixed and the JAR file would be searched from the root. Consult the java.lang.Class.getResource(String) method for more information.

In the example, a request for http://www.acme.com/files/myfile.html must map to the name "com/acme/www/myfile.html" which is in the bundle's JAR file.

More sophisticated implementations of the getResource(String) method could filter the input name, restricting the resources that may be returned or map the input name onto the file system (if the security implications of this action are acceptable).

Alternatively, the resource registration could have used a default HttpContext object, as demonstrated in the following call to registerResources:

getHttpService().registerResources(
    "files",
    "/com/acme/www",
    null
);

In this case, the Http Service implementation would call the createDefaultHttpContext() method and use its return value as the HttpContext argument for the registerResources method. The default implementation must map the resource request to the bundle's resource, using Bundle.getResource(String). In the case of the previous example, however, the internal name must now specify the full path to the directory containing the resource files in the JAR file. No automatic prefixing of the package name is done.

The getMimeType(String) implementation of the default HttpContext object should rely on the default mapping provided by the Http Service by returning null. Its handleSecurity(HttpServletRequest,HttpServletResponse) may implement an authentication mechanism that is implementation-dependent.

### 102.4 Mapping HTTP Requests to Servlet and Resource Registrations

When an HTTP request comes in from a client, the Http Service checks to see if the requested URI matches any registered aliases. A URI matches only if the path part of the URI is exactly the same string. Matching is case sensitive.

If it does match, a matching registration takes place, which is processed as follows:

1. If the registration corresponds to a servlet, the authorization is verified by calling the handleSecurity method of the associated HttpContext object. See Authentication on page 54. If the request is authorized, the servlet must be called by its service method to complete the HTTP request.
2. If the registration corresponds to a resource, the authorization is verified by calling the handleSecurity method of the associated HttpContext object. See Authentication on page 54. If the request is authorized, a target resource name is constructed from the requested URI by substituting the alias from the registration with the internal name from the registration if the alias is not "/". If the alias is "/", then the target resource name is constructed by prefixing the request-
ed URI with the internal name. An internal name of "/" is considered to have the value of the empty string ("" ) during this process.

3. The target resource name must be passed to the getRes ource method of the associated HttpSession object.

4. If the returned URL object is not null, the HttpSession must return the contents of the URL to the client completing the HTTP request. The translated target name, as opposed to the original requested URI, must also be used as the argument to HttpSession.getMimeType.

5. If the returned URL object is null, the HttpSession continues as if there was no match.

6. If there is no match, the HttpSession must attempt to match sub-strings of the requested URI to registered aliases. The sub-strings of the requested URI are selected by removing the last "/" and everything to the right of the last "/".

The HttpSession must repeat this process until either a match is found or the sub-string is an empty string. If the sub-string is empty and the alias "/" is registered, the request is considered to match the alias "/". Otherwise, the HttpSession must return HttpServletResponse.SC_NOT_FOUND (404) to the client.

For example, an HTTP request comes in with a request URI of "/fudd/bugs/foo.txt", and the only registered alias is "/fudd". A search for "/fudd/bugs/foo.txt" will not match an alias. Therefore, the HttpSession will search for the alias "/fudd/bugs" and the alias "/fudd". The latter search will result in a match and the matched alias registration must be used.

Registrations for identical aliases are not allowed. If a bundle registers the alias "/fudd", and another bundle tries to register the exactly the same alias, the second caller must receive a NamespaceException and its resource or servlet must not be registered. It could, however, register a similar alias - for example, "/fudd/bugs", as long as no other registration for this alias already exists.

The following table shows some examples of the usage of the name-space.

<table>
<thead>
<tr>
<th>Alias</th>
<th>Internal Name</th>
<th>URI</th>
<th>getRes ource Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>(empty string)</td>
<td>/fudd/bugs</td>
<td>/fudd/bugs</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
<td>/fudd/bugs</td>
<td>/fudd/bugs</td>
</tr>
<tr>
<td>/fudd</td>
<td>/tmp</td>
<td>/fudd/bugs</td>
<td>/tmp/fudd/bugs</td>
</tr>
<tr>
<td>/fudd</td>
<td>(empty string)</td>
<td>/fudd/bugs</td>
<td>/bugs</td>
</tr>
<tr>
<td>/fudd</td>
<td>/</td>
<td>/fudd/bugs</td>
<td>/fudd/bugs</td>
</tr>
<tr>
<td>/fudd</td>
<td>/tmp</td>
<td>/fudd/bugs</td>
<td>/tmp/fudd/bugs</td>
</tr>
<tr>
<td>/fudd</td>
<td>tmp</td>
<td>/fudd/bugs/x.gif</td>
<td>tmp/bugs</td>
</tr>
<tr>
<td>/fudd/bugs/x.gif</td>
<td>tmp/y.gif</td>
<td>/fudd/bugs/x.gif</td>
<td>tmp/y.gif</td>
</tr>
</tbody>
</table>

### 102.5 The Default HttpSession Object

The HttpSession object in the first example demonstrates simple implementations of the HttpSession interface methods. Alternatively, the example could have used a default HttpSession object, as demonstrated in the following call to registerServlet:

```java
getHttpService().registerServlet(
    "/servletAlias",
    myServlet,
    initparams,
    null
);```
In this case, the HttpService implementation must call createDefault HttpContext and use the return value as the HttpContext argument.

If the default HttpContext object, and thus the ServletContext object, is to be shared by multiple servlet registrations, the previous servlet registration example code needs to be changed to use the same default HttpContext object. This change is demonstrated in the next example:

```java
HttpContext defaultContext = getHttpService().createDefaultHttpContext();
getHttpService().registerServlet(
    "/servletAlias",
    myServlet,
    initparams,
    defaultContext
);

// defaultContext can be reused
// for further servlet registrations
```

102.6 Multipurpose Internet Mail Extension (MIME) Types

MIME defines an extensive set of headers and procedures to encode binary messages in US-ASCII mails. For an overview of all the related RFCs, consult [4] MIME Multipurpose Internet Mail Extension. An important aspect of this extension is the type (file format) mechanism of the binary messages. The type is defined by a string containing a general category (text, application, image, audio and video, multipart, and message) followed by a "/" and a specific media type, as in the example, "text/html" for HTML formatted text files. A MIME type string can be followed by additional specifiers by separating key=value pairs with a semicolon (;). These specifiers can be used, for example, to define character sets as follows:

```
text/plain ; charset=iso-8859-1
```

The Internet Assigned Number Authority (IANA) maintains a set of defined MIME media types. This list can be found at [5] Assigned MIME Media Types. MIME media types are extendable, and when any part of the type starts with the prefix "x-", it is assumed to be vendor-specific and can be used for testing. New types can be registered as described in [6] Registration Procedures for new MIME media types.

HTTP bases its media typing on the MIME RFCs. The "Content-Type" header should contain a MIME media type so that the browser can recognize the type and format the content correctly.

The source of the data must define the MIME media type for each transfer. Most operating systems do not support types for files, but use conventions based on file names, such as the last part of the file name after the last ".". This extension is then mapped to a media type.

Implementations of the HttpService should have a reasonable default of mapping common extensions to media types based on file extensions.

### Table 102.2 Sample Extension to MIME Media Mapping

<table>
<thead>
<tr>
<th>Extension</th>
<th>MIME media type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.jpg</td>
<td>image/jpeg</td>
<td>JPEG Files</td>
</tr>
<tr>
<td>.jpeg</td>
<td>image/jpeg</td>
<td>GIF Files</td>
</tr>
<tr>
<td>.gif</td>
<td>image/gif</td>
<td>GIF Files</td>
</tr>
<tr>
<td>.css</td>
<td>text/css</td>
<td>Cascading Style Sheet Files</td>
</tr>
<tr>
<td>.txt</td>
<td>text/plain</td>
<td>Text Files</td>
</tr>
</tbody>
</table>

Licensed under the OSGi Distribution and Feedback License, Version 2.0. Copyright © OSGi Alliance. All Rights Reserved.
Only the bundle developer, however, knows exactly which files have what media type. The `HttpContext` interface can therefore be used to map this knowledge to the media type. The `HttpContext` class has the following method for this: `getMimeType(String)`.

The implementation of this method should inspect the file name and use its internal knowledge to map this name to a MIME media type.

Simple implementations can extract the extension and look up this extension in a table.

Returning null from this method allows the Http Service implementation to use its default mapping mechanism.

## Authentication

The Http Service has separated the authentication and authorization of a request from the execution of the request. This separation allows bundles to use available Servlet sub-classes while still providing bundle specific authentication and authorization of the requests.

Prior to servicing each incoming request, the Http Service calls the `handleSecurity(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)` method on the `HttpContext` object that is associated with the request URI. This method controls whether the request is processed in the normal manner or an authentication error is returned.

If an implementation wants to authenticate the request, it can use the authentication mechanisms of HTTP. See [RFC 2617: HTTP Authentication: Basic and Digest Access Authentication](#). These mechanisms normally interpret the headers and decide if the user identity is available, and if it is, whether that user has authenticated itself correctly.

There are many different ways of authenticating users, and the `handleSecurity` method on the `HttpContext` object can use whatever method it requires. If the method returns true, the request must continue to be processed using the potentially modified `HttpServletRequest` and `HttpServletResponse` objects. If the method returns false, the request must not be processed.

A common standard for HTTP is the basic authentication scheme that is not secure when used with HTTP. Basic authentication passes the password in base 64 encoded strings that are trivial to decode into clear text. Secure transport protocols like HTTPS use SSL to hide this information. With these protocols basic authentication is secure.

Using basic authentication requires the following steps:

1. If no Authorization header is set in the request, the method should set the `WWW-Authenticate` header in the response. This header indicates the desired authentication mechanism and the realm. For example, `WWW-Authenticate: Basic realm="ACME"`.

   The header should be set with the response object that is given as a parameter to the `handleSecurity` method. The `handleSecurity` method should set the status to `HttpServletResponse.SC_UNAUTHORIZED (401)` and return false.

2. Secure connections can be verified with the `ServletRequest.getScheme()` method. This method returns, for example, "https" for an SSL connection; the `handleSecurity` method can use this and other information to decide if the connection's security level is acceptable. If not, the `handleSecurity` method should set the status to `HttpServletResponse.SC_FORBIDDEN (403)` and return false.
3. Next, the request must be authenticated. When basic authentication is used, the Authorization
header is available in the request and should be parsed to find the user and password. See [7] RFC
2617: HTTP Authentication: Basic and Digest Access Authentication for more information.

If the user cannot be authenticated, the status of the response object should be set to
HttpServletResponse.SC_UNAUTHORIZED (401) and return false.

4. The authentication mechanism that is actually used and the identity of the authenticated user
can be of interest to the Servlet object. Therefore, the implementation of the handleSecurity
method should set this information in the request object using the ServletRequest.setAttribute
method. This specification has defined a number of OSGi-specific attribute names for this pur-
pose:

- AUTHENTICATION_TYPE - Specifies the scheme used in authentication. A Servlet may re-
trieve the value of this attribute by calling the HttpServletRequest.getAuthType method.
  This attribute name is org.osgi.service.http.authentication.type.
- REMOTE_USER - Specifies the name of the authenticated user. A Servlet may retrieve the
  value of this attribute by calling the HttpServletRequest.getRemoteUser method. This at-
  tribute name is org.osgi.service.http.authentication.remote.user.
- AUTHORIZATION - If a User Admin service is available in the environment, then the
  handleSecurity method should set this attribute with the Authorization object ob-
tained from the User Admin service. Such an object encapsulates the authentica-
tion of its remote user. A Servlet may retrieve the value of this attribute by calling
  HttpServletRequest.getAttribute(HttpContext.AUTHORIZATION). This header name is
  org.osgi.service.useradmin.authorization.

5. Once the request is authenticated and any attributes are set, the handleSecurity method should
return true. This return indicates to the Http Service that the request is authorized and pro-
cessing may continue. If the request is for a Servlet, the Http Service must then call the service
method on the Servlet object.

102.8 Security

This section only applies when executing in an OSGi environment which is enforcing Java permis-
sions.

102.8.1 Accessing Resources with the Default HttpContext

The Http Service must be granted AdminPermission[*,RESOURCE] so that bundles may use a de-
fault HttpContext object. This is necessary because the implementation of the default HttpContext
object must call Bundle.getResource to access the resources of a bundle and this method requires
the caller to have AdminPermission[bundle,RESOURCE].

Any bundle may access resources in its own bundle by calling Class.getResource. This operation
is privileged. The resulting URL object may then be passed to the Http Service as the result of a
HttpContext.getResource call. No further permission checks are performed when accessing bundle
resource URL objects, so the Http Service does not need to be granted any additional permissions.

102.8.2 Accessing Other Types of Resources

In order to access resources that were not registered using the default HttpContext object, the Http
Service must be granted sufficient privileges to access these resources. For example, if the getResource
method of the registered HttpContext object returns a file URL, the Http Service requires the
 corresponding FilePermission to read the file. Similarly, if the getResource method of the registered
HttpContext object returns an HTTP URL, the Http Service requires the corresponding SocketPer-
mission to connect to the resource.
Configuration Properties

Therefore, in most cases, the Http Service should be a privileged service that is granted sufficient permission to serve any bundle's resources, no matter where these resources are located. Therefore, the Http Service must capture the AccessControlContext object of the bundle registering resources or a servlet, and then use the captured AccessControlContext object when accessing resources returned by the registered HttpContext object. This situation prevents a bundle from registering resources that it does not have permission to access.

Therefore, the Http Service should follow a scheme like the following example. When a resource or servlet is registered, it should capture the context.

```java
AccessControlContext acc = AccessController.getContext();
```

When a URL returned by the getURL method of the associated HttpContext object is called, the Http Service must call the getURL method in a doPrivileged construct using the AccessControlContext object of the registering bundle:

```java
AccessController.doPrivileged(
    new PrivilegedExceptionAction() {
        public Object run() throws Exception {
            ...
        }
    }, acc);
```

The Http Service must only use the captured AccessControlContext when accessing resource URL objects.

102.8.3 Servlet and HttpContext objects

This specification does not require that the Http Service is granted All Permission or wraps calls to the Servlet and HttpContext objects in a doPrivileged block. Therefore, it is the responsibility of the Servlet and HttpContext implementations to use a doPrivileged block when performing privileged operations.

102.9 Configuration Properties

If the Http Service does not have its port values configured through some other means, the Http Service implementation should use the following properties to determine the port values upon which to listen.

The following OSGi environment properties are used to specify default HTTP ports:

- `org.osgi.service.http.port` - This property specifies the port used for servlets and resources accessible via HTTP. The default value for this property is 80.
- `org.osgi.service.http.port.secure` - This property specifies the port used for servlets and resources accessible via HTTPS. The default value for this property is 443.

102.10 org.osgi.service.http

Http Service Package Version 1.2.

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.http; version="[1.2,2.0)"

Example import for providers implementing the API in this package:
Import-Package: org.osgi.service.http; version="[1.2,1.3)"

102.10.1 Summary
- HttpContext - This interface defines methods that the Http Service may call to get information about a registration.
- HttpService - The Http Service allows other bundles in the OSGi environment to dynamically register resources and servlets into the URI namespace of Http Service.
- NamespaceException - A NamespaceException is thrown to indicate an error with the caller's request to register a servlet or resources into the URI namespace of the Http Service.

102.10.2 public interface HttpContext
This interface defines methods that the Http Service may call to get information about a registration.
Servlets and resources may be registered with an HttpContext object; if no HttpContext object is specified, a default HttpContext object is used. Servlets that are registered using the same HttpContext object will share the same ServletContext object.
This interface is implemented by users of the HttpService.

102.10.2.1 public static final String AUTHENTICATION_TYPE = "org.osgi.service.http.authentication.type"
HttpServletRequest attribute specifying the scheme used in authentication. The value of the attribute can be retrieved by HttpServletRequest.getAuthType. This attribute name is org.osgi.service.http.authentication.type.
Since 1.1

102.10.2.2 public static final String AUTHORIZATION = "org.osgi.service.useradmin.authorization"
HttpServletRequest attribute specifying the Authorization object obtained from the org.osgi.service.useradmin.UserAdmin service. The value of the attribute can be retrieved by HttpServletRequest.getAttribute(HttpContext.AUTHORIZATION). This attribute name is org.osgi.service.useradmin.authorization.
Since 1.1

102.10.2.3 public static final String REMOTE_USER = "org.osgi.service.http.authentication.remote.user"
HttpServletRequest attribute specifying the name of the authenticated user. The value of the attribute can be retrieved by HttpServletRequest.getRemoteUser. This attribute name is org.osgi.service.http.authentication.remote.user.
Since 1.1

102.10.2.4 public String getMimeType(String name)
name determine the MIME type for this name.
- Maps a name to a MIME type. Called by the Http Service to determine the MIME type for the name. For servlet registrations, the Http Service will call this method to support the ServletContext method getMimeType. For resource registrations, the Http Service will call this method to determine the MIME type for the Content-Type header in the response.
Returns MIME type (e.g. text/html) of the name or null to indicate that the Http Service should determine the MIME type itself.
102.10.5  public URL getResource(String name)

name the name of the requested resource

Maps a resource name to a URL.

Called by the Http Service to map a resource name to a URL. For servlet registrations, Http Service will call this method to support the ServletContext methods getResource and getResourceAsStream. For resource registrations, Http Service will call this method to locate the named resource. The context can control from where resources come. For example, the resource can be mapped to a file in the bundle’s persistent storage area via bundleContext.getDataFile(name).toURL() or to a resource in the context’s bundle via getClass().getResource(name)

Returns URL that Http Service can use to read the resource or null if the resource does not exist.

102.10.6  public boolean handleSecurity(HttpServletRequest request,HttpServletResponse response) throws IOException

request the HTTP request
response the HTTP response

Handles security for the specified request.

The Http Service calls this method prior to servicing the specified request. This method controls whether the request is processed in the normal manner or an error is returned.

If the request requires authentication and the Authorization header in the request is missing or not acceptable, then this method should set the WWW-Authenticate header in the response object, set the status in the response object to Unauthorized(401) and return false. See also RFC 2617: HTTP Authentication: Basic and Digest Access Authentication (available at http://www.ietf.org/rfc/rfc2617.txt).

If the request requires a secure connection and the getScheme method in the request does not return 'https' or some other acceptable secure protocol, then this method should set the status in the response object to Forbidden(403) and return false.

When this method returns false, the Http Service will send the response back to the client, thereby completing the request. When this method returns true, the Http Service will proceed with servicing the request.

If the specified request has been authenticated, this method must set the AUTHENTICATION_TYPE request attribute to the type of authentication used, and the REMOTE_USER request attribute to the remote user (request attributes are set using the setAttribute method on the request). If this method does not perform any authentication, it must not set these attributes.

If the authenticated user is also authorized to access certain resources, this method must set the AUTHORIZATION request attribute to the Authorization object obtained from the org.osgi.service.useradmin.UserAdmin service.

The servlet responsible for servicing the specified request determines the authentication type and remote user by calling the getAuthType and getRemoteUser methods, respectively, on the request.

Returns true if the request should be serviced, false if the request should not be serviced and Http Service will send the response back to the client.

Throws IOException – may be thrown by this method. If this occurs, the Http Service will terminate the request and close the socket.

102.10.3  public interface HttpService

The Http Service allows other bundles in the OSGi environment to dynamically register resources and servlets into the URI namespace of Http Service. A bundle may later unregister its resources or servlets.
See Also  HttpContext

No Implement  Consumers of this API must not implement this interface

102.10.3.1  

public HttpContext createDefaultHttpContext()

□  Creates a default HttpContext for registering servlets or resources with the HttpService, a new Http-Context object is created each time this method is called.

The behavior of the methods on the default HttpContext is defined as follows:

•  getMimeType - Does not define any customized MIME types for the Content-Type header in the response, and always returns null.
•  handleSecurity - Performs implementation-defined authentication on the request.
•  getResource - Assumes the named resource is in the context bundle; this method calls the context bundle's Bundle.getResource method, and returns the appropriate URL to access the resource. On a Java runtime environment that supports permissions, the Http Service needs to be granted org.osgi.framework.AdminPermission[*,,RESOURCE].

Returns  a default HttpContext object.

Since  1.1

102.10.3.2  

public void registerResources(String alias,String name,HttpContext context) throws NamespaceException

alias  name in the URI namespace at which the resources are registered

name  the base name of the resources that will be registered

context  the HttpContext object for the registered resources, or null if a default HttpContext is to be created and used.

□  Registers resources into the URI namespace.

The alias is the name in the URI namespace of the Http Service at which the registration will be mapped. An alias must begin with slash ('/') and must not end with slash ('/'), with the exception that an alias of the form "/" is used to denote the root alias. The name parameter must also not end with slash ('/') with the exception that a name of the form "/" is used to denote the root of the bundle. See the specification text for details on how HTTP requests are mapped to servlet and resource registrations.

For example, suppose the resource name /tmp is registered to the alias /files. A request for /files/ foo.txt will map to the resource name /tmp/foo.txt.

httpservice.registerResources("/files", "/tmp", context);

The Http Service will call the HttpContext argument to map resource names to URLs and MIME types and to handle security for requests. If the HttpContext argument is null, a default HttpContext is used (see createDefaultHttpContext()).

Throws  NamespaceException – if the registration fails because the alias is already in use.

IllegalArgumentException – if any of the parameters are invalid

102.10.3.3  

public void registerServlet(String alias,Servlet servlet,Dictionary initparams,HttpContext context) throws ServletException, NamespaceException

alias  name in the URI namespace at which the servlet is registered

servlet  the servlet object to register

initparams  initialization arguments for the servlet or null if there are none. This argument is used by the servlet's ServletConfig object.
Registers a servlet into the URI namespace.

The alias is the name in the URI namespace of the Http Service at which the registration will be mapped.

An alias must begin with slash ('/') and must not end with slash ('/'), with the exception that an alias of the form '/' is used to denote the root alias. See the specification text for details on how HTTP requests are mapped to servlet and resource registrations.

The Http Service will call the servlet's init method before returning.

```java
httpService.registerServlet("/myservlet", servlet, initparams, context);
```

Servlets registered with the same HttpContext object will share the same ServletContext.

The Http Service will call the context argument to support the ServletContext methods `getResource`, `getResourceAsStream` and `getMimeType`, and to handle security for requests. If the context argument is null, a default HttpContext object is used (see createDefaultHttpContext()).

**Throws**

- `NamespaceException` – if the registration fails because the alias is already in use.
- `javax.servlet.ServletException` – if the servlet's init method throws an exception, or the given servlet object has already been registered at a different alias.
- `IllegalArgumentException` – if any of the arguments are invalid

### 102.10.4 `public void unregister(String alias)`

**alias** name in the URI name-space of the registration to unregister

Unregisters a previous registration done by registerServlet or registerResources methods.

After this call, the registered alias in the URI name-space will no longer be available. If the registration was for a servlet, the Http Service must call the destroy method of the servlet before returning.

If the bundle which performed the registration is stopped or otherwise "unget's" the Http Service without calling unregister(String) then Http Service must automatically unregister the registration. However, if the registration was for a servlet, the destroy method of the servlet will not be called in this case since the bundle may be stopped. unregister(String) must be explicitly called to cause the destroy method of the servlet to be called. This can be done in the BundleActivator.stop method of the bundle registering the servlet.

**Throws**

- `IllegalArgumentException` – if there is no registration for the alias or the calling bundle was not the bundle which registered the alias.

### 102.10.4.1 `public NamespaceException(String message)`

**message** the detail message

Construct a NamespaceException object with a detail message.

### 102.10.4.2 `public NamespaceException(String message, Throwable cause)`

**message** The detail message.

**cause** The nested exception.
Construct a NamespaceException object with a detail message and a nested exception.

102.10.4.3 public Throwable getCause()

- Returns the cause of this exception or null if no cause was set.

Returns: The cause of this exception or null if no cause was set.

Since: 1.2

102.10.4.4 public Throwable getException()

- Returns the nested exception.

This method predates the general purpose exception chaining mechanism. The getCause() method is now the preferred means of obtaining this information.

Returns: The result of calling getCause().

102.10.4.5 public Throwable initCause(Throwable cause)

cause: The cause of this exception.

- Initializes the cause of this exception to the specified value.

Returns: This exception.

Throws: IllegalArgumentException—If the specified cause is this exception.

IllegalStateException—If the cause of this exception has already been set.

Since: 1.2

102.11 References

[1] HTTP 1.0 Specification RFC-1945

[2] HTTP 1.1 Specification RFC-2616


[4] MIME Multipurpose Internet Mail Extension
http://www.mhonarc.org/~ehood/MIME/MIME.html

[5] Assigned MIME Media Types
http://www.iana.org/assignments/media-types

[6] Registration Procedures for new MIME media types
http://www.ietf.org/rfc/rfc2048.txt

http://www.ietf.org/rfc/rfc2617.txt
104 Configuration Admin Service Specification

Version 1.5

104.1 Introduction

The Configuration Admin service is an important aspect of the deployment of an OSGi framework. It allows an Operator to configure deployed bundles. Configuring is the process of defining the configuration data for bundles and assuring that those bundles receive that data when they are active in the OSGi framework.

Figure 104.1 Configuration Admin Service Overview

104.1.1 Essentials

The following requirements and patterns are associated with the Configuration Admin service specification:

- **Local Configuration**: The Configuration Admin service must support bundles that have their own user interface to change their configurations.
- **Reflection**: The Configuration Admin service must be able to deduce the names and types of the needed configuration data.
- **Legacy**: The Configuration Admin service must support configuration data of existing entities (such as devices).
- **Object Oriented**: The Configuration Admin service must support the creation and deletion of instances of configuration information so that a bundle can create the appropriate number of services under the control of the Configuration Admin service.
- **Embedded Devices**: The Configuration Admin service must be deployable on a wide range of platforms. This requirement means that the interface should not assume file storage on the platform. The choice to use file storage should be left to the implementation of the Configuration Admin service.
Remote versus Local Management - The Configuration Admin service must allow for a remotely managed OSGi framework, and must not assume that configuration information is stored locally. Nor should it assume that the Configuration Admin service is always done remotely. Both implementation approaches should be viable.

Availability - The OSGi environment is a dynamic environment that must run continuously (24/7/365). Configuration updates must happen dynamically and should not require restarting of the system or bundles.

Immediate Response - Changes in configuration should be reflected immediately.

Execution Environment - The Configuration Admin service will not require more than an environment that fulfills the minimal execution requirements.

Communications - The Configuration Admin service should not assume “always-on” connectivity, so the API is also applicable for mobile applications in cars, phones, or boats.

Extendability - The Configuration Admin service should expose the process of configuration to other bundles. This exposure should at a minimum encompass initiating an update, removing certain configuration properties, adding properties, and modifying the value of properties potentially based on existing property or service values.

Complexity Trade-offs - Bundles in need of configuration data should have a simple way of obtaining it. Most bundles have this need and the code to accept this data. Additionally, updates should be simple from the perspective of the receiver.

Trade-offs in simplicity should be made at the expense of the bundle implementing the Configuration Admin service and in favor of bundles that need configuration information. The reason for this choice is that normal bundles will outnumber Configuration Admin bundles.

Regions - It should be possible to create groups of bundles and a manager in a single system that share configuration data that is not accessible outside the region.

Shared Information - It should be possible to share configuration data between bundles.

### 104.1.2 Entities

- **Configuration information** - The information needed by a bundle before it can provide its intended functionality.

- **Configuration dictionary** - The configuration information when it is passed to the target service. It consists of a Dictionary object with a number of properties and identifiers.

- **Configuring Bundle** - A bundle that modifies the configuration information through the Configuration Admin service. This bundle is either a management bundle or the bundle for which the configuration information is intended.

- **Configuration Target** - The target service that will receive the configuration information. For services, there are two types of targets: ManagedServiceFactory or ManagedService objects.

- **Configuration Admin Service** - This service is responsible for supplying configuration target bundles with their configuration information. It maintains a database with configuration information, keyed on the service.pid of configuration target services. These services receive their configuration dictionary/dictionaries when they are registered with the Framework. Configurations can be modified or extended using Configuration Plugin services before they reach the target bundle.

- **Managed Service** - A Managed Service represents a client of the Configuration Admin service, and is thus a configuration target. Bundles should register a Managed Service to receive the configuration data from the Configuration Admin service. A Managed Service adds one or more unique service.pid service properties as a primary key for the configuration information.

- **Managed Service Factory** - A Managed Service Factory can receive a number of configuration dictionaries from the Configuration Admin service, and is thus also a configuration target service. It should register with one or more service.pid strings and receives zero or more configuration dictionaries. Each dictionary has its own PID that is distinct from the factory PID.
- **Configuration Object** - Implements the Configuration interface and contains the configuration dictionary for a Managed Service or one of the configuration dictionaries for a Managed Service Factory. These objects are manipulated by configuring bundles.

- **Configuration Plugin Services** - Configuration Plugin services are called before the configuration dictionary is given to the configuration targets. The plug-in can modify the configuration dictionary, which is passed to the Configuration Target.

**Figure 104.2 Overall Service Diagram**

**104.1.3 Synopsis**

This specification is based on the concept of a Configuration Admin service that manages the configuration of an OSGi framework. It maintains a database of Configuration objects, locally or remotely. This service monitors the service registry and provides configuration information to services that are registered with a service.pid property, the Persistent IDentity (PID), and implement one of the following interfaces:

- **Managed Service** - A service registered with this interface receives its configuration dictionary from the database or receives null when no such configuration exists.

- **Managed Service Factory** - Services registered with this interface can receive several configuration dictionaries when registered. The database contains zero or more configuration dictionaries for this service. Each configuration dictionary is given sequentially to the service.

The database can be manipulated either by the Management Agent or bundles that configure themselves. Other parties can provide Configuration Plugin services. Such services participate in the configuration process. They can inspect the configuration dictionary and modify it before it reaches the target service.

**104.2 Configuration Targets**

One of the more complicated aspects of this specification is the subtle distinction between the ManagedService and ManagedServiceFactory classes. Both receive configuration information from the Configuration Admin service and are treated similarly in most respects. Therefore, this specification refers to configuration targets or simply targets when the distinction is irrelevant.

The difference between these types is related to the cardinality of the configuration dictionary. A Managed Service is used when an existing entity needs a configuration dictionary. Thus, a one-to-one relationship always exists between the configuration dictionary and the configurable entity in the Managed Service. There can be multiple Managed Service targets registered with the same PID but a Managed Service can only configure a single entity in each given Managed Service.
A Managed Service Factory is used when part of the configuration is to define *how many instances are required* for a given Managed Service Factory. A management bundle can create, modify, and delete any number of instances for a Managed Service Factory through the Configuration Admin service. Each instance is configured by a single Configuration object. Therefore, a Managed Service Factory can have multiple associated Configuration objects.

A Configuration target updates the target when the underlying Configuration object is created, updated, or deleted. However, it is not called back when the Configuration Admin service is shutdown or the service is ungotten.

To summarize:

- A *Managed Service* must receive a single configuration dictionary when it is registered or when its configuration is modified.
- A *Managed Service Factory* must receive from zero to *n* configuration dictionaries when it registers, depending on the current configuration. The Managed Service Factory is informed of configuration dictionary changes: modifications, creations, and deletions.

### 104.3 The Persistent Identity

A crucial concept in the Configuration Admin service specification is the Persistent IDentity (PID) as defined in the Framework’s service layer. Its purpose is to act as a primary key for objects that need a configuration dictionary. The name of the service property for PID is defined in the Framework in `org.osgi.framework.Constants.SERVICE_PID`.

The Configuration Admin service requires the use of one or more PIDs with Managed Service and Managed Service Factory registrations because it associates its configuration data with PIDs.

A service can register with multiple PIDs and PIDs can be shared between multiple targets (both Managed Service and Managed Service Factory targets) to receive the same information. If PIDs are to be shared between Bundles then the location of the Configuration must be a multi-location, see *Location Binding* on page 69.

The Configuration Admin must track the configuration targets on their actual PID. That is, if the `service.pid` service property is modified then the Configuration Admin must treat it as if the service was unregistered and then re-registered with the new PID.

#### 104.3.1 PID Syntax

PIDs are intended for use by other bundles, not by people, but sometimes the user is confronted with a PID. For example, when installing an alarm system, the user needs to identify the different components to a wiring application. This type of application exposes the PID to end users.

PIDs should follow the symbolic-name syntax, which uses a very restricted character set. The following sections define some schemes for common cases. These schemes are not required, but bundle developers are urged to use them to achieve consistency.
104.3.1.1 Local Bundle PIDs

As a convention, descriptions starting with the bundle identity and a full stop (‘.’) are reserved for a bundle. As an example, a PID of “65.536” would belong to the bundle with a bundle identity of 65.

104.3.1.2 Software PIDs

Configuration target services that are singletons can use a Java package name they own as the PID (the reverse domain name scheme) as long as they do not use characters outside the basic ASCII set. As an example, the PID named com.acme.watchdog would represent a Watchdog service from the ACME company.

104.3.1.3 Devices

Devices are usually organized on buses or networks. The identity of a device, such as a unique serial number or an address, is a good component of a PID. The format of the serial number should be the same as that printed on the housing or box, to aid in recognition.

Table 104.1 Schemes for Device-Oriented PID Names

<table>
<thead>
<tr>
<th>Bus</th>
<th>Example</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB</td>
<td>USB.0123-0002-9909873</td>
<td>idVendor (hex 4) &lt;br&gt; idProduct (hex 4) &lt;br&gt; iSerialNumber (decimal)</td>
<td>Universal Serial Bus. Use the standard device descriptor.</td>
</tr>
<tr>
<td>IP</td>
<td>IP.172.16.28.21</td>
<td>IP nr (dotted decimal)</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>802</td>
<td>802-00:60:97:00:9A:56</td>
<td>MAC address with : separators</td>
<td>IEEE 802 MAC address (Token Ring, Ethernet,...)</td>
</tr>
<tr>
<td>ONE</td>
<td>ONE.06-00000021E461</td>
<td>Family (hex 2) and serial number including CRC (hex 6)</td>
<td>1-wire bus of Dallas Semiconductor</td>
</tr>
<tr>
<td>COM</td>
<td>COM.krups-brewer-12323</td>
<td>serial number or type name of device</td>
<td>Serial ports</td>
</tr>
</tbody>
</table>

104.3.2 Targeted PIDs

PIDs are defined as primary keys for the configuration object; any target that uses the PID in its service registration (and has the proper permissions if security is on) will receive the configuration associated with it, regardless of the bundle that registered the target service. Though in general the PID is designed to ignore the bundle, there are a number of cases where the bundle becomes relevant. The most typical case is where a bundle is available in different versions. Each version will request the same PID and will get therefore configured identically.

Targeted PIDs are specially formatted PIDs that are interpreted by the Configuration Admin service. Targeted PIDs work both as a normal Managed Service PID and as a Managed Service Factory PID. In the case of factories, the targeted PID is the Factory PID since the other PID is chosen by CM for each instance.

The target PID scopes the applicability of the PID to a limited set of target bundles. The syntax of a target pid is:

```
target-pid ::= PID <br> ( '|' symbolic-name ( '|' version ( '|' location )?) )? )?
```

Targets never register with a target PID, target PIDs should only be used when creating, getting, or deleting a Configuration through the Configuration Admin service. The target PID is still the primary key of the Configuration and is thus in itself a PID. The distinction is only made when the Configuration Admin must update a target service. Instead of using the non-target PID as the primary key it must first search if there exists a target PID in the Configuration store that matches the requested target PID.
When a target registers and needs to be updated the Configuration Admin must first find the Configuration with the best matching PID. It must logically take the requested PID, append it with the bundle symbolic name, the bundle version, and the bundle location. The version must be formatted canonically, that is, according to the toString() method of the Version class. The rules for best matching are then as follows:

Look for a Configuration, in the given order, with a key of:

- `<pid>|<bsn>|<version>|<location>`
- `<pid>|<bsn>|<version>`
- `<pid>|<bsn>`
- `<pid>`

For example:

- `com.example.web.WebConf|com.acme.example|3.2.0`
- `com.example.web.WebConf|com.acme.example`
- `com.example.web.WebConf`

If a registered target service has a PID that contains a vertical line ('|') then the value must be taken as is and must not be interpreted as a targeted PID.

The service.pid configuration property for a targeted PID configuration must always be set to the targeted PID. That is, if the PID is `com.example.web.WebConf` and the targeted PID is `com.example.web.WebConf|com.acme.example|3.2.0` then the property in the Configuration dictionary must be the targeted PID.

If a Configuration with a targeted PID is deleted or a Configuration with a new targeted PID is added then all targets that would be stale must be reevaluated against the new situation and updated accordingly if they are no longer bound against the best matching target PID.

### 104.3.3 Extenders and Targeted PIDs

Extenders like Declarative Services use Configurations but bypass the general Managed Service or Managed Service Factory method. It is the responsibility of these extenders to access the Configurations using the targeted PIDs.

Since getting a Configuration tends to create that Configuration it is necessary for these extenders to use the listConfigurations(String) method to find out if a more targeted Configuration exists.

There are many ways the extender can find the most targeted PID. For example, the following code gets the most targeted PID for a given bundle.

```java
String mostTargeted(String pid, Bundle bundle,String key) throws Exception {
    String p = escape(pid); // make sure does not control filter
    String bsn = bundle.getSymbolicName();
    String version = bundle.getVersion().toString();
    String location = bundle.getLocation();
    String f = String.format(
        "(|(%s=%s)(%s=%s|%s)(%s=%s|%s|%s)(%s=%s|%s|%s|%s))", //
        key, p, //
        key, p, bsn, //
        key, p, bsn, version, //
        key, p, bsn, version, location );

    Configuration[] configurations = cm.listConfigurations(f);
    if (configurations == null)
        return null;
```
The Configuration Object

104.4 The Configuration Object

A Configuration object contains the configuration dictionary, which is a set of properties that configure an aspect of a bundle. A bundle can receive Configuration objects by registering a configuration target service with a PID service property. See The Persistent Identity on page 66 for more information about PIDs.

During registration, the Configuration Admin service must detect these configuration target services and hand over their configuration dictionary via a callback. If this configuration dictionary is subsequently modified, the modified dictionary is handed over to the configuration target with the same callback.

The Configuration object is primarily a set of properties that can be updated by a Management Agent, user interfaces on the OSGi framework, or other applications. Configuration changes are first made persistent, and then passed to the target service via a call to the updated method in the ManagedServiceFactory or ManagedService class.

A Configuration object must be uniquely bound to a Managed Service or Managed Service Factory. This implies that a bundle must not register a Managed Service Factory with a PID that is the same as the PID given to a Managed Service.

104.4.1 Location Binding

When a Configuration object is created with either getConfiguration(String) or createFactoryConfiguration(String), it becomes bound to the location of the calling bundle. This location is obtained with the getBundleLocation() method.

Location binding is a security feature that assures that only management bundles can modify configuration data, and other bundles can only modify their own configuration data. A Security Exception is thrown if a bundle does not have ConfigurationPermission[location, CONFIGURE].

The two argument versions of getConfiguration(String, String) and createFactoryConfiguration(String, String) take a location String as their second argument. These methods require the correct permission, and they create Configuration objects bound to the specified location.

Locations can be specified for a specific Bundle or use multi-locations. For a specific location the Configuration location must exactly match the location of the target’s Bundle. A multi-location is any location that has the following syntax:

multi-location ::= '?' symbolic-name?

For example

?com.acme
The path after the question mark is the multi-location name, the multi-location name can be empty if only a question mark is specified. Configurations with a multi-location are dispatched to any target that has visibility to the Configuration. The visibility for a given Configuration c depends on the following rules:

- **Single-Location** - If c.location is not a multi-location then a Bundle only has visibility if the Bundle's location exactly matches c.location. In this case there is never a security check.

- **Multi-Location** - If c.location is a multi-location (that is, starts with a question mark):
  - **Security Off** - The Bundle always has visibility
  - **Security On** - The target's Bundle must have ConfigurationPermission[ c.location, TARGET ] as defined by the Bundle's hasPermission method. The resource name of the permission must include the question mark.

The permission matches on the whole name, including any leading ?. The TARGET action is only applicable in the multi-location scenario since the security is not checked for a single-location. There is therefore no point in granting a Bundle a permission with TARGET action for anything but a multi-location (starting with a ?).

It is therefore possible to register services with the same PID from different bundles. If a multi-location is used then each bundle will be evaluated for a corresponding configuration update. If the bundle has visibility then it is updated, otherwise it is not.

If multiple targets must be updated then the order of updating is the ranking order of their services.

If a target loses visibility because the Configuration's location changes then it must immediately be deleted from the perspective of that target. That is, the target must see a deletion (Managed Service Factory) or an update with null (Managed Service). If a configuration target gains visibility then the target must see a new update with the proper configuration dictionary. However, the associated events must not be sent as the underlying Configuration is not actually deleted nor modified.

Changes in the permissions must not initiate a recalculation of the visibility. If the permissions are changed this will not become visible until one of the other events happen that cause a recalculation of the visibility.

If the location is changed then the Configuration Admin must send a CM_LOCATION_CHANGED event to signal that the location has changed. It is up to the Configuration Listeners to update their state appropriately.

### 104.4.2 Dynamic Binding

Dynamic binding is available for backward compatibility with earlier versions. It is recommended that management agents explicitly set the location to a ? (a multi-location) to allow multiple bundles to share PIDs and not use the dynamic binding facility. If a management agent uses ?, it must at least have ConfigurationPermission[ ?, CONFIGURE ] when security is on, it is also possible to use ConfigurationPermission[ ?*, CONFIGURE ] to not limit the management agent. See Regions on page 82 for some examples of using the locations in isolation scenarios.

A null location parameter can be used to create Configuration objects that are not yet bound. In this case, the Configuration becomes bound to a specific location the first time that it is compared to a Bundle's location. If a bundle becomes dynamically bound to a Configuration then a CM_LOCATION_CHANGED event must be dispatched.

When this dynamically bound Bundle is subsequently uninstalled, configurations that are bound to this bundle must be released. That means that for such Configuration object's the bundle location must be set to null again so it can be bound again to another bundle.
104.4.3 Configuration Properties

A configuration dictionary contains a set of properties in a Dictionary object. The value of the property must be the same type as the set of Primary Property Types specified in OSGi Core Release 6 Filter Syntax.

The name or key of a property must always be a String object, and is not case sensitive during look up, but must preserve the original case. The format of a property name should be:

property-name ::= public | private
public ::= symbolic-name // See General Syntax in Core Framework
private ::= '.' symbolic-name

Properties can be used in other subsystems that have restrictions on the character set that can be used. The symbolic-name production uses a very minimal character set.

Bundles must not use nested vectors or arrays, nor must they use mixed types. Using mixed types or nesting makes it impossible to use the metatype specification. See Metatype Service Specification on page 107.

Property values that are collections may have an ordering that must be preserved when persisting the configuration so that later access to the property value will see the preserved ordering of the collection.

104.4.4 Property Propagation

A configuration target should copy the public configuration properties (properties whose name does not start with a ‘.’ or ‘u002E’) of the Dictionary object argument in updated(Dictionary) into the service properties on any resulting service registration.

This propagation allows the development of applications that leverage the Framework service registry more extensively, so compliance with this mechanism is advised.

A configuration target may ignore any configuration properties it does not recognize, or it may change the values of the configuration properties before these properties are registered as service properties. Configuration properties in the Framework service registry are not strictly related to the configuration information.

Bundles that follow this recommendation to propagate public configuration properties can participate in horizontal applications. For example, an application that maintains physical location information in the Framework service registry could find out where a particular device is located in the house or car. This service could use a property dedicated to the physical location and provide functions that leverage this property, such as a graphic user interface that displays these locations.

Bundles performing service registrations on behalf of other bundles (e.g. OSGi Declarative Services) should propagate all public configuration properties and not propagate private configuration properties.

104.4.5 Automatic Properties

The Configuration Admin service must automatically add a number of properties to the configuration dictionary. If these properties are also set by a configuring bundle or a plug-in, they must always be overridden before they are given to the target service, see Configuration Plugin on page 85. Therefore, the receiving bundle or plug-in can assume that the following properties are defined by the Configuration Admin service and not by the configuring bundle:

- service.pid - Set to the PID of the associated Configuration object. This is the full the targeted PID if a targeted PID is used, see Targeted PIDs on page 67.
- service.factoryPid - Only set for a Managed Service Factory. It is then set to the PID of the associated Managed Service Factory. This is the full the targeted PID if a targeted PID is used.
Managed Service

- `service.bundleLocation` - Set to the location of the Configuration object. This property can only be used for searching, it may not appear in the configuration dictionary returned from the `getProperties` method due to security reasons, nor may it be used when the target is updated.

Constants for some of these properties can be found in `org.osgi.framework.Constants` and the `ConfigurationAdmin` interface. These service properties are all of type `String`.

104.4.6 Equality

Two different Configuration objects can actually represent the same underlying configuration. This means that a Configuration object must implement the `equals` and `hashCode` methods in such a way that two Configuration objects are equal when their PID is equal.

104.5 Managed Service

A Managed Service is used by a bundle that needs one or more configuration dictionaries. It therefore registers the Managed Service with one or more PIDs and is thus associated with one Configuration object in the Configuration Admin service for each registered PID. A bundle can register any number of ManagedService objects, but each must be identified with its own PID or PIDs.

A bundle should use a Managed Service when it needs configuration information for the following:

- **A Singleton** - A single entity in the bundle that needs to be configured.
- **Externally Detected Devices** - Each device that is detected causes a registration of an associated ManagedService object. The PID of this object is related to the identity of the device, such as the address or serial number.

A Managed Service may be registered with more than one PID and therefore be associated with multiple Configuration objects, one for each PID. Using multiple PIDs for a Managed Service is not recommended. For example, when a configuration is deleted for a Managed Service there is no way to identify which PID is associated with the deleted configuration.

104.5.1 Singletons

When an object must be instantiated only once, it is called a *singleton*. A singleton requires a single configuration dictionary. Bundles may implement several different types of singletons if necessary.

For example, a Watchdog service could watch the registry for the status and presence of services in the Framework service registry. Only one instance of a Watchdog service is needed, so only a single configuration dictionary is required that contains the polling time and the list of services to watch.

104.5.2 Networks

When a device in the external world needs to be represented in the OSGi Environment, it must be detected in some manner. The Configuration Admin service cannot know the identity and the number of instances of the device without assistance. When a device is detected, it still needs configuration information in order to play a useful role.

For example, a 1-Wire network can automatically detect devices that are attached and removed. When it detects a temperature sensor, it could register a Sensor service with the Framework service registry. This Sensor service needs configuration information specifically for that sensor, such as which lamps should be turned on, at what temperature the sensor is triggered, what timer should be started, in what zone it resides, and so on. One bundle could potentially have hundreds of these sensors and actuators, and each needs its own configuration information.

Each of these Sensor services should be registered as a Managed Service with a PID related to the physical sensor (such as the address) to receive configuration information.
Other examples are services discovered on networks with protocols like Jini, UPnP, and Salutation. They can usually be represented in the Framework service registry. A network printer, for example, could be detected via UPnP. Once in the service registry, these services usually require local configuration information. A Printer service needs to be configured for its local role: location, access list, and so on.

This information needs to be available in the Framework service registry whenever that particular Printer service is registered. Therefore, the Configuration Admin service must remember the configuration information for this Printer service.

This type of service should register with the Framework as a Managed Service in order to receive appropriate configuration information.

**104.5.3 Configuring Managed Services**

A bundle that needs configuration information should register one or more `ManagedService` objects with a PID service property. If it has a default set of properties for its configuration, it may include them as service properties of the Managed Service. These properties may be used as a configuration template when a `Configuration` object is created for the first time. A Managed Service optionally implements the `MetaTypeProvider` interface to provide information about the property types. See `MetaTyping` on page 87.

When this registration is detected by the Configuration Admin service, the following steps must occur:

- The configuration stored for the registered PID must be retrieved. If there is a `Configuration` object for this PID and the configuration is visible for the associated bundle then it is sent to the Managed Service with `updated(Dictionary)`.
- If a Managed Service is registered and no configuration information is available or the configuration is not visible then the Configuration Admin service must call `updated(Dictionary)` with a null parameter.
- If the Configuration Admin service starts after a Managed Service is registered, it must call `updated(Dictionary)` on this service as soon as possible according to the prior rules. For this reason, a Managed Service must always get a callback when it registers and the Configuration Admin service is started.

Multiple Managed Services can register with the same PID, they are all updated as long as they have visibility to the configuration as defined by the location, see `Location Binding` on page 69.

The `updated(Dictionary)` callback from the Configuration Admin service to the Managed Service must take place asynchronously. This requirement allows the Managed Service to finish its initialization in a synchronized method without interference from the Configuration Admin service callback. Care should be taken not to cause deadlocks by calling the Framework within a synchronized method.
The updated method may throw a `ConfigurationException`. This object must describe the problem and what property caused the exception.

### 104.5.4 Race Conditions

When a Managed Service is registered, the default properties may be visible in the service registry for a short period before they are replaced by the properties of the actual configuration dictionary. Care should be taken that this visibility does not cause race conditions for other bundles.

In cases where race conditions could be harmful, the Managed Service must be split into two pieces: an object performing the actual service and a Managed Service. First, the Managed Service is registered, the configuration is received, and the actual service object is registered. In such cases, the use of a Managed Service Factory that performs this function should be considered.

### 104.5.5 Examples of Managed Service

Figure 104.5 shows a Managed Service configuration example. Two services are registered under the ManagedService interface, each with a different PID.

The Configuration Admin service has a database containing a configuration record for each PID. When the Managed Service with `service.pid = com.acme` is registered, the Configuration Admin service will retrieve the properties `name=Elmer` and `size=42` from its database. The properties are stored in a Dictionary object and then given to the Managed Service with the `updated(Dictionary)` method.
104.5.5.1 Configuring A Console Bundle

In this example, a bundle can run a single debugging console over a Telnet connection. It is a singleton, so it uses a ManagedService object to get its configuration information: the port and the network name on which it should register.

```java
class SampleManagedService implements ManagedService{
    Dictionary properties; 
    ServiceRegistration registration;
    Console console;

    public void start(BundleContext context) throws Exception {
        properties = new Hashtable();
        properties.put( Constants.SERVICE_PID,
                         "com.acme.console ");

        registration = context.registerService(
            ManagedService.class.getName(),
            this, 
            properties 
        );
    }

    public synchronized void updated(Dictionary np) {
        if (np!= null) {
            properties = np;
            properties.put( 
                Constants.SERVICE_PID, "com.acme.console");
        }

        if (console == null)
            console = new Console();

        int port = ((Integer)properties.get("port")).intValue();
        String network = (String) properties.get("network");
        console.setPort(port, network);
        registration.setProperties(properties);
    }

    ... further methods
}
```

104.5.6 Deletion

When a Configuration object for a Managed Service is deleted, the Configuration Admin service must call updated(Dictionary) with a null argument on a thread that is different from that on which the Configuration.delete was executed. This deletion must send out a Configuration Event CM_DELETED asynchronously to any registered Configuration Listener services after the updated method is called with a null.
104.6 Managed Service Factory

A Managed Service Factory is used when configuration information is needed for a service that can be instantiated multiple times. When a Managed Service Factory is registered with the Framework, the Configuration Admin service consults its database and calls updated(String,Dictionary) for each associated and visible Configuration object that matches the PIDs on the registration. It passes the identifier of the Configuration instance, which can be used as a PID, as well as a Dictionary object with the configuration properties.

A Managed Service Factory is useful when the bundle can provide functionality a number of times, each time with different configuration dictionaries. In this situation, the Managed Service Factory acts like a class and the Configuration Admin service can use this Managed Service Factory to instantiate instances for that class.

In the next section, the word factory refers to this concept of creating instances of a function defined by a bundle that registers a Managed Service Factory.

104.6.1 When to Use a Managed Service Factory

A Managed Service Factory should be used when a bundle does not have an internal or external entity associated with the configuration information but can potentially be instantiated multiple times.

104.6.1.1 Example Email Fetcher

An email fetcher program displays the number of emails that a user has - a function likely to be required for different users. This function could be viewed as a class that needs to be instantiated for each user. Each instance requires different parameters, including password, host, protocol, user id, and so on.

An implementation of the Email Fetcher service should register a ManagedServiceFactory object. In this way, the Configuration Admin service can define the configuration information for each user separately. The Email Fetcher service will only receive a configuration dictionary for each required instance (user).

104.6.1.2 Example Temperature Conversion Service

Assume a bundle has the code to implement a conversion service that receives a temperature and, depending on settings, can turn an actuator on and off. This service would need to be instantiated many times depending on where it is needed. Each instance would require its own configuration information for the following:

- Upper value
- Lower value
- Switch Identification
- ...

Such a conversion service should register a service object under a ManagedServiceFactory interface. A configuration program can then use this Managed Service Factory to create instances as needed. For example, this program could use a Graphic User Interface (GUI) to create such a component and configure it.

104.6.1.3 Serial Ports

Serial ports cannot always be used by the OSGi Device Access specification implementations. Some environments have no means to identify available serial ports, and a device on a serial port cannot always provide information about its type.

Therefore, each serial port requires a description of the device that is connected. The bundle managing the serial ports would need to instantiate a number of serial ports under the control of the Con-
The Configuration Admin service, with the appropriate DEVICECATEGORY property to allow it to participate in the Device Access implementation.

If the bundle cannot detect the available serial ports automatically, it should register a Managed Service Factory. The Configuration Admin service can then, with the help of a configuration program, define configuration information for each available serial port.

### 104.6.2 Registration

Similar to the Managed Service configuration dictionary, the configuration dictionary for a Managed Service Factory is identified by a PID. The Managed Service Factory, however, also has a factory PID, which is the PID of the associated Managed Service Factory. It is used to group all Managed Service Factory configuration dictionaries together.

When a Configuration object for a Managed Service Factory is created (ConfigurationAdmin.createFactoryConfiguration(String,String) or createFactoryConfiguration(String)), a new unique PID is created for this object by the Configuration Admin service. The scheme used for this PID is defined by the Configuration Admin service and is unrelated to the factory PID, which is chosen by the registering bundle.

When the Configuration Admin service detects the registration of a Managed Service Factory, it must find all visible configuration dictionaries for this factory and must then sequentially call ManagedServiceFactory.updated(String,Dictionary) for each configuration dictionary. The first argument is the PID of the Configuration object (the one created by the Configuration Admin service) and the second argument contains the configuration properties.

The Managed Service Factory should then create any artifacts associated with that factory. Using the PID given in the Configuration object, the bundle may register new services (other than a Managed Service) with the Framework, but this is not required. This may be necessary when the PID is useful in contexts other than the Configuration Admin service.

The receiver must not register a Managed Service with this PID because this would force two Configuration objects to have the same PID. If a bundle attempts to do this, the Configuration Admin service should log an error and must ignore the registration of the Managed Service.

The Configuration Admin service must guarantee that no race conditions exist between initialization, updates, and deletions.

![Managed Service Factory Action Diagram](image)

A Managed Service Factory has only one update method: updated(String,Dictionary). This method can be called any number of times as Configuration objects are created or updated.

The Managed Service Factory must detect whether a PID is being used for the first time, in which case it should create a new instance, or a subsequent time, in which case it should update an existing instance.
The Configuration Admin service must call `updated(String, Dictionary)` on a thread that is different from the one that executed the registration. This requirement allows an implementation of a Managed Service Factory to use a synchronized method to assure that the callbacks do not interfere with the Managed Service Factory registration.

The `updated(String, Dictionary)` method may throw a `ConfigurationException` object. This object describes the problem and what property caused the problem. These exceptions should be logged by a Configuration Admin service.

Multiple Managed Service Factory services can be registered with the same PID. Each of those services that have visibility to the corresponding configuration will be updated in service ranking order.

104.6.3 Deletion

If a configuring bundle deletes an instance of a Managed Service Factory, the `deleted(String)` method is called. The argument is the PID for this instance. The implementation of the Managed Service Factory must remove all information and stop any behavior associated with that PID. If a service was registered for this PID, it should be unregistered.

Deletion will asynchronously send out a Configuration Event `CM_DELETED` to all registered Configuration Listener services.

104.6.4 Managed Service Factory Example

Figure 104.7 highlights the differences between a Managed Service and a Managed Service Factory. It shows how a Managed Service Factory implementation receives configuration information that was created before it was registered.

- A bundle implements an EMail Fetcher service. It registers a `ManagedServiceFactory` object with PID `com.acme.email`.
- The Configuration Admin service notices the registration and consults its database. It finds three `Configuration` objects for which the factory PID is equal to `com.acme.email`. It must call `updated(String, Dictionary)` for each of these `Configuration` objects on the newly registered `ManagedServiceFactory` object.
- For each configuration dictionary received, the factory should create a new instance of an EMailFetcher object, one for erica (PID=16.1), one for anna (PID=16.3), and one for elmer (PID=16.2).
- The EMailFetcher objects are registered under the Topic interface so their results can be viewed by an online display.

If the EMailFetcher object is registered, it may safely use the PID of the Configuration object because the Configuration Admin service must guarantee its suitability for this purpose.
104.6.5 Multiple Consoles Example

This example illustrates how multiple consoles, each of which has its own port and interface can run simultaneously. This approach is very similar to the example for the Managed Service, but highlights the difference by allowing multiple consoles to be created.

class ExampleFactory implements ManagedServiceFactory{
    Hashtable consoles = new Hashtable();
    BundleContext context;

    public void start( BundleContext context )
        throws Exception {
        this.context = context;
        Hashtable local = new Hashtable();
        local.put(Constants.SERVICE_PID, "com.acme.console");
        context.registerService(
            ManagedServiceFactory.class.getName(),
            this,
            local );
    }

    public void updated( String pid, Dictionary config ){
        Console console = (Console) consoles.get(pid);
        if (console == null) {
            console = new Console(context);
            consoles.put(pid, console);
        }

        int port = getInt(config, "port", 2011);
        String network = getString(
            config,
            "network",
            null /*all*/
        );
        console.setPort(port, network);
    }
}
104.7 Configuration Admin Service

The ConfigurationAdmin interface provides methods to maintain configuration data in an OSGi environment. This configuration information is defined by a number of Configuration objects associated with specific configuration targets. Configuration objects can be created, listed, modified, and deleted through this interface. Either a remote management system or the bundles configuring their own configuration information may perform these operations.

The ConfigurationAdmin interface has methods for creating and accessing Configuration objects for a Managed Service, as well as methods for managing new Configuration objects for a Managed Service Factory.

104.7.1 Creating a Managed Service Configuration Object

A bundle can create a new Managed Service Configuration object with ConfigurationAdmin.getConfiguration. No create method is offered because doing so could introduce race conditions between different bundles trying to create a Configuration object for the same Managed Service. The getConfiguration method must atomically create and persistently store an object if it does not yet exist.

Two variants of this method are:

- getConfiguration(String) - This method is used by a bundle with a given location to configure its own ManagedService objects. The argument specifies the PID of the targeted service.
- getConfiguration(String,String) - This method is used by a management bundle to configure another bundle. Therefore, this management bundle needs the right permission. The first argument is the PID and the second argument is the location identifier of the targeted ManagedService object.

All Configuration objects have a method, getFactoryPid(), which in this case must return null because the Configuration object is associated with a Managed Service.

Creating a new Configuration object must not initiate a callback to the Managed Service updated method until the properties are set in the Configuration with the update method.

104.7.2 Creating a Managed Service Factory Configuration Object

The ConfigurationAdmin class provides two methods to create a new instance of a Managed Service Factory:

- createFactoryConfiguration(String) - This method is used by a bundle with a given location to configure its own ManagedServiceFactory objects. The argument specifies the PID of the target ManagedServiceFactory object. This factory PID can be obtained from the returned Configuration object with the getFactoryPid() method.
- createFactoryConfiguration(String,String) - This method is used by a management bundle to configure another bundle's ManagedServiceFactory object. The first argument is the PID and the
second is the location identifier of the targeted ManagedServiceFactory object. The factory PID can be obtained from the returned Configuration object with getFactoryPid method.

Creating a new factory configuration must not initiate a callback to the Managed Service Factory updated method until the properties are set in the Configuration object with the update method.

104.7.3 Accessing Existing Configurations

The existing set of Configuration objects can be listed with listConfigurations(String). The argument is a String object with a filter expression. This filter expression has the same syntax as the Framework Filter class. For example:

\( (&(size=42)(service.factoryPid=*osgi*)) \)

The Configuration Admin service must only return Configurations that are visible to the calling bundle, see Location Binding on page 69.

A single Configuration object is identified with a PID, and can be obtained with listConfigurations(String) if it is visible. null is returned in both cases when there are no visible Configuration objects.

The PIDs that are filtered on can be targeted PIDs, see Targeted PIDs on page 67.

104.7.4 Updating a Configuration

The process of updating a Configuration object is the same for Managed Services and Managed Service Factories. First, listConfigurations(String) or getConfiguration(String) should be used to get a Configuration object. The properties can be obtained with Configuration.getProperties. When no update has occurred since this object was created, getProperties returns null.

New properties can be set by calling Configuration.update. The Configuration Admin service must first store the configuration information and then call all configuration targets that have visibility with the updated method: either the ManagedService.updated(Dictionary) or ManagedServiceFactory.updated(String,Dictionary) method. If a target service is not registered, the fresh configuration information must be given to the target when the configuration target service registers and it has visibility. Each update of the Configuration properties must update a counter in the Configuration object after the data has been persisted but before the target(s) have been updated and any events are sent out. This counter is available from the getChangeCount() method.

The update method calls in Configuration objects are not executed synchronously with the related target services updated method. The updated method must be called asynchronously. The Configuration Admin service, however, must have updated the persistent storage before the update method returns.

The update method must also asynchronously send out a Configuration Event CM_UPDATED to all registered Configuration Listeners.

104.7.5 Using Multi-locations

Sharing configuration between different bundles can be done using multi-locations, see Location Binding on page 69. A multi-location for a Configuration enables this Configuration to be delivered to any bundle that has visibility to that configuration. It is also possible that Bundles are interested in multiple PIDs for one target service, for this reason they can register multiple PIDs for one service.

For example, a number of bundles require access to the URL of a remote host, associated with the PID com.acme.host. A manager, aware that this PID is used by different bundles, would need to specify a location for the Configuration that allows delivery to any bundle. A multi-location, any location starting with a question mark achieves this. The part after the question mark has only use if the system runs with security, it allows the implementation of regions, see Regions on page 82. In
this example a single question mark is used because any Bundle can receive this Configuration. The manager's code could look like:

```java
Configuration c = admin.getConfiguration("com.acme.host", "?" );
Hashtable ht = new Hashtable();
ht.put("host", hostURL);
c.update(ht);
```

A Bundle interested in the host configuration would register a Managed Service with the following properties:

- `service.pid = [ "com.acme.host", "com.acme.system"]`

The Bundle would be called back for both the `com.acme.host` and `com.acme.system` PID and must therefore discriminate between these two cases. This Managed Service therefore would have a callback like:

```java
volatile URL url;
public void updated( Dictionary d ) {
    if ( d.get("service.pid").equals("com.acme.host") )
        this.url = new URL( d.get("host") );
    if ( d.get("service.pid").equals("com.acme.system") )
        ....
}
```

### 104.7.6 Regions

In certain cases it is necessary to isolate bundles from each other. This will require that the configuration can be separated in *regions*. Each region can then be configured by a separate manager that is only allowed to manage bundles in its own region. Bundles can then only see configurations from their own region. Such a region based system can only be achieved with Java security as this is the only way to place bundles in a sandbox. This section describes how the Configuration's location binding can be used to implement regions if Java security is active.

Regions are groups of bundles that share location information among each other but are not willing to share this information with others. Using the multi-locations, see Location Binding on page 69, and security it is possible to limit access to a Configuration by using a location name. A Bundle can only receive a Configuration when it has `ConfigurationPermission[location name, TARGET]`. It is therefore possible to create region by choosing a region name for the location. A management agent then requires `ConfigurationPermission[?region-name, CONFIGURE]` and a Bundle in the region requires `ConfigurationPermission[?region-name, TARGET]`.

To implement regions, the management agent is required to use multi-locations; without the question mark a Configuration is only visible to a Bundle that has the exact location of the Configuration. With a multi-location, the Configuration is delivered to any bundle that has the appropriate permission. Therefore, if regions are used, no manager should have `ConfigurationPermission[*, CONFIGURE]` because it would be able to configure anybody. This permission would enable the manager to set the location to any region or set the location to null. All managers must be restricted to a permission like `ConfigurationPermission[?com.acme.region.*,CONFIGURE]`. The resource name for a Configuration Permission uses substring matching as in the OSGi Filter, this facility can be used to simplify the administrative setup and implement more complex sharing schemes.

For example, a management agent works for the region `com.acme`. It has the following permission:

```java
ConfigurationPermission[?com.acme.*.*,CONFIGURE]
```

The manager requires multi-location updates for `com.acme.*` (the last full stop is required in this wildcarding). For the `CONFIGURE` action the question mark must be specified in the resource name. The bundles in the region have the permission:
ConfigurationPermission["?com.acme.alpha", TARGET]

The question mark must be specified for the TARGET permission. A management agent that needs to
configure Bundles in a region must then do this as follows:

Configuration c = admin.getConfiguration("com.acme.host", "?com.acme.alpha" );
Hashtable ht = new Hashtable();
ht.put("host", hostURL);
c.update(ht);

Another, similar, example with two regions:

- system
- application

There is only one manager that manages all bundles. Its permissions look like:

ConfigurationPermission[?system, CONFIGURE]
ConfigurationPermission[?application, CONFIGURE]

A Bundle in the application region can have the following permissions:

ConfigurationPermission[?application, TARGET]

This managed bundle therefore has only visibility to configurations in the application region.

104.7.7 Deletion

A Configuration object that is no longer needed can be deleted with Configuration.delete, which
removes the Configuration object from the database. The database must be updated before the target
service's updated or deleted method is called. Only services that have received the configuration
dictionary before must be called.

If the target service is a Managed Service Factory, the factory is informed of the deleted Configuration
object by a call to ManagedServiceFactory.deleted(String) method. It should then remove the
associated instance. The ManagedServiceFactory.deleted(String) call must be done asynchronously
with respect to Configuration.delete().

When a Configuration object of a Managed Service is deleted, ManagedService.updated is called
with null for the properties argument. This method may be used for clean-up, to revert to default
values, or to unregister a service. This method is called asynchronously from the delete method.

The update method must also asynchronously send out a Configuration Event CM_DELETED to all
registered Configuration Listeners.

104.7.8 Updating a Bundle's Own Configuration

The Configuration Admin service specification does not distinguish between updates via a Management
Agent and a bundle updating its own configuration information (as defined by its location). Even if a bundle updates its own configuration information, the Configuration Admin service must
callback the associated target service's updated method.

As a rule, to update its own configuration, a bundle's user interface should only update the configuration information and never its internal structures directly. This rule has the advantage that the
events, from the bundle implementation's perspective, appear similar for internal updates, remote
management updates, and initialization.
104.8 Configuration Events

Configuration Admin can update interested parties of changes in its repository. The model is based on the white board pattern where Configuration Listener services are registered with the service registry.

There are two types of Configuration Listener services:

- **ConfigurationListener** - The default Configuration Listener receives events asynchronously from the method that initiated the event and on another thread.
- **SynchronousConfigurationListener** - A Synchronous Configuration Listener is guaranteed to be called on the same thread as the method call that initiated the event.

The Configuration Listener service will receive **ConfigurationException** objects if important changes take place. The Configuration Admin service must call the `configurationEvent(ConfigurationEvent)` method with such an event. Configuration Events must be delivered in order for each listener as they are generated. The way events must be delivered is the same as described in Delivering Events of OSGi Core Release 6.

The **ConfigurationException** object carries a factory PID (**getFactoryPid()**) and a PID (**getPid()**). If the factory PID is null, the event is related to a Managed Service Configuration object, else the event is related to a Managed Service Factory Configuration object.

The **ConfigurationException** object can deliver the following events from the **getType()** method:

- **CM_DELETED** - The Configuration object is deleted.
- **CM_UPDATED** - The Configuration object is updated.
- **CM_LOCATION_CHANGED** - The location of the Configuration object changed.

The Configuration Event also carries the ServiceReference object of the Configuration Admin service that generated the event.

104.8.1 Event Admin Service and Configuration Change Events

Configuration events must be delivered asynchronously by the Configuration Admin implementation, if present. The topic of a configuration event must be:

```
org.osgi.service.cm/ConfigurationException/<eventtype>
```

The `<event type>` can be any of the following:

```
CM_DELETED
CM_UPDATED
CM_LOCATION_CHANGED
```

The properties of a configuration event are:

- **cm.factoryPid** - (String) The factory PID of the associated Configuration object, if the target is a Managed Service Factory. Otherwise not set.
- **cm.pid** - (String) The PID of the associated Configuration object.
- **service** - (ServiceReference) The Service Reference of the Configuration Admin service.
- **service.id** - (Long) The Configuration Admin service's ID.
- **service.objectClass** - (String[]) The Configuration Admin service's object class (which must include org.osgi.service.cm.ConfigurationAdmin)
- **service.pid** - (String) The Configuration Admin service's persistent identity, if set.
104.9 Configuration Plugin

The Configuration Admin service allows third-party applications to participate in the configuration process. Bundles that register a service object under a ConfigurationPlugin interface can process the configuration dictionary just before it reaches the configuration target service.

Plug-ins allow sufficiently privileged bundles to intercept configuration dictionaries just before they must be passed to the intended Managed Service or Managed Service Factory but after the properties are stored. The changes the plug-in makes are dynamic and must not be stored. The plug-in must only be called when an update takes place while it is registered and there is a valid dictionary. The plug-in is not called when a configuration is deleted.

The ConfigurationPlugin interface has only one method: `modifyConfiguration(ServiceReference, Dictionary)`. This method inspects or modifies configuration data.

All plug-ins in the service registry must be traversed and called before the properties are passed to the configuration target service. Each Configuration Plugin object gets a chance to inspect the existing data, look at the target object, which can be a ManagedService object or a ManagedServiceFactory object, and modify the properties of the configuration dictionary. The changes made by a plug-in must be visible to plugins that are called later.

ConfigurationPlugin objects should not modify properties that belong to the configuration properties of the target service unless the implications are understood. This functionality is mainly intended to provide functions that leverage the Framework service registry. The changes made by the plug-in should normally not be validated. However, the Configuration Admin must ignore changes to the automatic properties as described in Automatic Properties on page 71.

For example, a Configuration Plugin service may add a physical location property to a service. This property can be leveraged by applications that want to know where a service is physically located. This scenario could be carried out without any further support of the service itself, except for the general requirement that the service should propagate the public properties it receives from the Configuration Admin service to the service registry.

### 104.9.1 Limiting The Targets

A ConfigurationPlugin object may optionally specify a cm.target registration property. This value is the PID of the configuration target whose configuration updates the ConfigurationPlugin object wants to intercept.

The ConfigurationPlugin object must then only be called with updates for the configuration target service with the specified PID. For a factory target service, the factory PID is used and the plugin will see all instances of the factory. Omitting the cm.target registration property means that it is called for all configuration updates.
### 104.9.2 Example of Property Expansion

Consider a Managed Service that has a configuration property `service.to` with the value `(objectclass=com.acme.Alarm)`. When the Configuration Admin service sets this property on the target service, a ConfigurationPlugin object may replace the `(objectclass=com.acme.Alarm)` filter with an array of existing alarm systems' PIDs as follows:

```
ID "service.to=[32434, 232, 12421, 1212]"
```

A new Alarm Service with `service.pid=343` is registered, requiring that the list of the target service be updated. The bundle which registered the Configuration Plugin service, therefore, wants to set the `service.to` registration property on the target service. It does **not** do this by calling `ManagedService.updated` directly for several reasons:

- In a securely configured system, it should not have the permission to make this call or even obtain the target service.
- It could get into race conditions with the Configuration Admin service if it had the permissions in the previous bullet. Both services would compete for access simultaneously.

Instead, it must get the `Configuration` object from the Configuration Admin service and call the `update` method on it.

The Configuration Admin service must schedule a new update cycle on another thread, and sometime in the future must call `ConfigurationPlugin.modifyProperties`. The `ConfigurationPlugin` object could then set the `service.to` property to `[32434, 232, 12421, 1212, 343]`. After that, the Configuration Admin service must call `updated` on the target service with the new `service.to` list.

### 104.9.3 Configuration Data Modifications

Modifications to the configuration dictionary are still under the control of the Configuration Admin service, which must determine whether to accept the changes, hide critical variables, or deny the changes for other reasons.

The `ConfigurationPlugin` interface must also allow plugins to detect configuration updates to the service via the callback. This ability allows them to synchronize the configuration updates with transient information.

### 104.9.4 Forcing a Callback

If a bundle needs to force a Configuration Plugin service to be called again, it must fetch the appropriate `Configuration` object from the Configuration Admin service and call the `update()` method (the no parameter version) on this object. This call forces an update with the current configuration dictionary so that all applicable plugins get called again.

### 104.9.5 Calling Order

The order in which the `ConfigurationPlugin` objects are called must depend on the `service.cmRanking` configuration property of the `ConfigurationPlugin` object. Table 104.2 shows the usage of the `service.cmRanking` property for the order of calling the Configuration Plugin services.

<table>
<thead>
<tr>
<th><code>service.cmRanking</code> value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>The Configuration Plugin service should not modify properties and must be called before any modifications are made.</td>
</tr>
<tr>
<td>( \geq 0 ) &amp;&amp; ( \leq 1000 )</td>
<td>The Configuration Plugin service modifies the configuration data. The calling order should be based on the value of the <code>service.cmRanking</code> property.</td>
</tr>
<tr>
<td>&gt; 1000</td>
<td>The Configuration Plugin service should not modify data and is called after all modifications are made.</td>
</tr>
</tbody>
</table>
104.10 Meta Typing

This section discusses how the Metatype specification is used in the context of a Configuration Admin service.

When a Managed Service or Managed Service Factory is registered, the service object may also implement the MetaTypeProvider interface.

If the Managed Service or Managed Service Factory object implements the MetaTypeProvider interface, a management bundle may assume that the associated ObjectClassDefinition object can be used to configure the service.

The ObjectClassDefinition and AttributeDefinition objects contain sufficient information to automatically build simple user interfaces. They can also be used to augment dedicated interfaces with accurate validations.

When the Metatype specification is used, care should be taken to match the capabilities of the metatype package to the capabilities of the Configuration Admin service specification. Specifically:

- The metatype specification cannot describe nested arrays and vectors or arrays/vectors of mixed type.

This specification does not address how the metatype is made available to a management system due to the many open issues regarding remote management.

104.11 Security

104.11.1 Configuration Permission

Every bundle has the implicit right to receive and configure configurations with a location that exactly matches the Bundle’s location or that is null. For all other situations the Configuration Admin must verify that the configuring and to be updated bundles have a Configuration Permission that matches the Configuration’s location.

The resource name of this permission maps to the location of the Configuration, the location can control the visibility of a Configuration for a bundle. The resource name is compared with the actual configuration location using the OSGi Filter sub-string matching. The question mark for multi-locations is part of the given resource name. The Configure Permission has the following actions:

- **CONFIGURE** - Can manage matching configurations
- **TARGET** - Can be updated with a matching configuration

To be able to set the location to null requires a ConfigurationPermission[*, CONFIGURE].

It is possible to deny bundles the use of multi-locations by using Conditional Permission Admin's deny model.

104.11.2 Permissions Summary

Configuration Admin service security is implemented using Service Permission and Configuration Permission. The following table summarizes the permissions needed by the Configuration Admin bundle itself, as well as the typical permissions needed by the bundles with which it interacts.

Configuration Admin:

ServicePermission[ ..ConfigurationAdmin, REGISTER]
ServicePermission[ ..ManagedService, GET ]
ServicePermission[ ..ManagedServiceFactory, GET ]
ServicePermission[ ..ConfigurationPlugin, GET ]
ConfigurationPermission[ * , CONFIGURE ]
AdminPermission[ * , METADATA ]

Managed Service:

ServicePermission[ ..ConfigurationAdmin, GET ]
ServicePermission[ ..ManagedService, REGISTER ]
ConfigurationPermission[ ... , TARGET ]

Managed Service Factory:

ServicePermission[ ..ConfigurationAdmin, GET ]
ServicePermission[ ..ManagedServiceFactory, REGISTER ]
ConfigurationPermission[ ... , TARGET ]

Configuration Plugin:

ServicePermission[ ..ConfigurationPlugin, REGISTER ]

Configuration Listener:

ServicePermission[ ..ConfigurationListener, REGISTER ]

The Configuration Admin service must have ServicePermission[ ConfigurationAdmin, REGISTER ]. It will also be the only bundle that needs the ServicePermission[ManagedService | ManagedServiceFactory | ConfigurationPlugin, GET]. No other bundle should be allowed to have GET permission for these interfaces. The Configuration Admin bundle must also hold ConfigurationPermission[* , CONFIGURE].

Bundles that can be configured must have the ServicePermission[ManagedService | ManagedServiceFactory, REGISTER]. Bundles registering ConfigurationPlugin objects must have ServicePermission[ConfigurationPlugin, REGISTER]. The Configuration Admin service must trust all services registered with the ConfigurationPlugin interface. Only the Configuration Admin service should have ServicePermission[ ConfigurationPlugin, GET ].

If a Managed Service or Managed Service Factory is implemented by an object that is also registered under another interface, it is possible, although inappropriate, for a bundle other than the Configuration Admin service implementation to call the updated method. Security-aware bundles can avoid this problem by having their updated methods check that the caller has ConfigurationPermission[* , CONFIGURE].

Bundles that want to change their own configuration need ServicePermission[ConfigurationAdmin, GET]. A bundle with ConfigurationPermission[* , CONFIGURE] is allowed to access and modify any Configuration object.

Pre-configuration of bundles requires ConfigurationPermission[location, CONFIGURE] (location can use the sub-string matching rules of the Filter) because the methods that specify a location require this permission.

## 104.11.3 Configuration and Permission Administration

Configuration information has a direct influence on the permissions needed by a bundle. For example, when the Configuration Admin Bundle orders a bundle to use port 2011 for a console, that bundle also needs permission for listening to incoming connections on that port.

Both a simple and a complex solution exist for this situation.

The simple solution for this situation provides the bundle with a set of permissions that do not define specific values but allow a range of values. For example, a bundle could listen to ports above 1024 freely. All these ports could then be used for configuration.
The other solution is more complicated. In an environment where there is very strong security, the bundle would only be allowed access to a specific port. This situation requires an atomic update of both the configuration data and the permissions. If this update was not atomic, a potential security hole would exist during the period of time that the set of permissions did not match the configuration.

The following scenario can be used to update a configuration and the security permissions:

1. Stop the bundle.
2. Update the appropriate Configuration object via the Configuration Admin service.
3. Update the permissions in the Framework.
4. Start the bundle.

This scenario would achieve atomicity from the point of view of the bundle.

104.12 org.osgi.service.cm

Configuration Admin Package Version 1.5.

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.cm; version="[1.5,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.service.cm; version="[1.5,1.6)"

104.12.1 Summary

• Configuration - The configuration information for a ManagedService or ManagedServiceFactory object.
• ConfigurationAdmin - Service for administering configuration data.
• ConfigurationEvent - A Configuration Event.
•ConfigurationException - An Exception class to inform the Configuration Admin service of problems with configuration data.
• ConfigurationListener - Listener for Configuration Events.
• ConfigurationPermission - Indicates a bundle’s authority to configure bundles or be updated by Configuration Admin.
• ConfigurationPlugin - A service interface for processing configuration dictionary before the update.
• ManagedService - A service that can receive configuration data from a Configuration Admin service.
• ManagedServiceFactory - Manage multiple service instances.
• SynchronousConfigurationListener - Synchronous Listener for Configuration Events.

104.12.2 Permissions

104.12.2.1 Configuration

• setBundleLocation(String)
  • ConfigurationPermission[this.location,CONFIGURE] - if this.location is not null
104.12.2 ConfigurationAdmin

- createFactoryConfiguration(String, String)
  - ConfigurationPermission[location, CONFIGURE] - if location is not null
  - ConfigurationPermission[* , CONFIGURE] - if location is null or if location is null
- getConfiguration(String, String)
  - ConfigurationPermission[* , CONFIGURE] - if location is null or if the returned configuration c already exists and c.location is null
  - ConfigurationPermission[location, CONFIGURE] - if location is not null
  - ConfigurationPermission[c.location, CONFIGURE] - if the returned configuration c already exists and c.location is not null
- getConfiguration(String)
  - ConfigurationPermission[c.location, CONFIGURE] - If the configuration c already exists and c.location is not null
- listConfigurations(String)
  - ConfigurationPermission[c.location, CONFIGURE] - Only configurations c are returned for which the caller has this permission

104.12.3 ManagedService

- updated(Dictionary)
  - ConfigurationPermission[c.location, TARGET] - Required by the bundle that registered this service

104.12.4 ManagedServiceFactory

- updated(String, Dictionary)
  - ConfigurationPermission[c.location, TARGET] - Required by the bundle that registered this service

104.12.3 public interface Configuration

The configuration information for a ManagedService or ManagedServiceFactory object. The Configuration Admin service uses this interface to represent the configuration information for a ManagedService or for a service instance of a ManagedServiceFactory.

A Configuration object contains a configuration dictionary and allows the properties to be updated via this object. Bundles wishing to receive configuration dictionaries do not need to use this class - they register a ManagedService or ManagedServiceFactory. Only administrative bundles, and bundles wishing to update their own configurations need to use this class.

The properties handled in this configuration have case insensitive String objects as keys. However, case must be preserved from the last set key/value.

A configuration can be bound to a specific bundle or to a region of bundles using the location. In its simplest form the location is the location of the target bundle that registered a ManagedService or a Managed Service Factory. However, if the location starts with ? then the location indicates multiple delivery. In such a case the configuration must be delivered to all targets. If security is on, the Configuration Permission can be used to restrict the targets that receive updates. The Configuration Admin must only update a target when the configuration location matches the lo-
cation of the target's bundle or the target bundle has a Configuration Permission with the action ConfigurationPermission.TARGET and a name that matches the configuration location. The name in the permission may contain wildcards ('*') to match the location using the same substring matching rules as Filter. Bundles can always create, manipulate, and be updated from configurations that have a location that matches their bundle location.

If a configuration's location is null, it is not yet bound to a location. It will become bound to the location of the first bundle that registers a ManagedService or ManagedServiceFactory object with the corresponding PID.

The same Configuration object is used for configuring both a Managed Service Factory and a Managed Service. When it is important to differentiate between these two the term 'factory configuration' is used.

Concurrency Thread-safe
Provider Type Consumers of this API must not implement this type

104.12.3.1 public void delete() throws IOException

- Delete this Configuration object. Removes this configuration object from the persistent store. Notify asynchronously the corresponding Managed Service or Managed Service Factory. A ManagedService object is notified by a call to its updated method with a null properties argument. A ManagedServiceFactory object is notified by a call to its deleted method.

Also notifies all Configuration Listeners with a ConfigurationEvent.CM_DELETED event.

Throws
- IOException – If delete fails.
- IllegalStateException – If this configuration has been deleted.

104.12.3.2 public boolean equals(Object other)

- Configuration object to compare against

- Equality is defined to have equal PIDs Two Configuration objects are equal when their PIDs are equal.

Returns true if equal, false if not a Configuration object or one with a different PID.

104.12.3.3 public String getBundleLocation()

- Get the bundle location. Returns the bundle location or region to which this configuration is bound, or null if it is not yet bound to a bundle location or region. If the location starts with '?' then the configuration is delivered to all targets and not restricted to a single bundle.

Returns location to which this configuration is bound, or null.

Throws
- IllegalStateException – If this configuration has been deleted.
- SecurityException – when the required permissions are not available

Security ConfigurationPermission["this.location,CONFIGURE"] – if this.location is not null
ConfigurationPermission['*',CONFIGURE] – if this.location is null

104.12.3.4 public long getChangeCount()

- Get the change count. Each Configuration must maintain a change counter that is incremented with a positive value every time the configuration is updated and its properties are stored. The counter must be incremented before the targets are updated and events are sent out.

Returns A monotonically increasing value reflecting changes in this Configuration.

Throws
- IllegalStateException – If this configuration has been deleted.

Since 1.5
104.12.3.5  **public String getFactoryPid()**

- For a factory configuration return the PID of the corresponding Managed Service Factory, else return null.

*Returns*  factory PID or null

*Throws*  `IllegalStateException`– If this configuration has been deleted.

104.12.3.6  **public String getPid()**

- Get the PID for this Configuration object.

*Returns*  the PID for this Configuration object.

*Throws*  `IllegalStateException`– if this configuration has been deleted

104.12.3.7  **public Dictionary<String,Object> getProperties()**

- Return the properties of this Configuration object. The Dictionary object returned is a private copy for the caller and may be changed without influencing the stored configuration. The keys in the returned dictionary are case insensitive and are always of type String.

If called just after the configuration is created and before update has been called, this method returns null.

*Returns*  A private copy of the properties for the caller or null. These properties must not contain the "service.bundleLocation" property. The value of this property may be obtained from the `getBundleLocation()` method.

*Throws*  `IllegalStateException`– If this configuration has been deleted.

104.12.3.8  **public int hashCode()**

- Hash code is based on PID. The hash code for two Configuration objects must be the same when the Configuration PID’s are the same.

*Returns*  hash code for this Configuration object

104.12.3.9  **public void setBundleLocation(String location)**

- Bind this Configuration object to the specified location. If the location parameter is null then the Configuration object will not be bound to a location/region. It will be set to the bundle’s location before the first time a Managed Service/Managed Service Factory receives this Configuration object via the updated method and before any plugins are called. The bundle location or region will be set persistently.

If the location starts with ? then all targets registered with the given PID must be updated.

If the location is changed then existing targets must be informed. If they can no longer see this configuration, the configuration must be deleted or updated with null. If this configuration becomes visible then they must be updated with this configuration.

Also notifies all Configuration Listeners with a `ConfigurationEvent.CM_LOCATION_CHANGED` event.

*Throws*  `IllegalStateException`– If this configuration has been deleted.

SecurityException– when the required permissions are not available

SecurityException– when the required permissions are not available

Security  `ConfigurationPermission[location,CONFIGURE]` – if location is not null

`ConfigurationPermission[this.location,CONFIGURE]` – if this.location is not null

`ConfigurationPermission["*",CONFIGURE]` – if this.location is null or if location is null
**104.12.3.10**

```java
public void update(Dictionary<String,?> properties) throws IOException
```

- **properties**

  the new set of properties for this configuration

  □ Update the properties of this Configuration object. Stores the properties in persistent storage after adding or overwriting the following properties:

  - “service.pid” : is set to be the PID of this configuration.
  - “service.factoryPid” : if this is a factory configuration it is set to the factory PID else it is not set.

  These system properties are all of type String.

  If the corresponding Managed Service/Managed Service Factory is registered, its updated method must be called asynchronously. Else, this callback is delayed until aforementioned registration occurs.

  Also notifies all Configuration Listeners with a ConfigurationEvent.CM_UPDATED event.

  **Throws**

  - IOException – if update cannot be made persistent
  - IllegalArgumentException – if the Dictionary object contains invalid configuration types or contains case variants of the same key name.
  - IllegalStateException – If this configuration has been deleted.

**104.12.3.11**

```java
public void update() throws IOException
```

- □ Update the Configuration object with the current properties. Initiate the updated callback to the Managed Service or Managed Service Factory with the current properties asynchronously.

  This is the only way for a bundle that uses a Configuration Plugin service to initiate a callback. For example, when that bundle detects a change that requires an update of the Managed Service or Managed Service Factory via its ConfigurationPlugin object.

  **Throws**

  - IOException – if update cannot access the properties in persistent storage
  - IllegalStateException – If this configuration has been deleted.

**See Also**

ConfigurationPlugin

**104.12.4**

**public interface ConfigurationAdmin**

Service for administering configuration data.

The main purpose of this interface is to store bundle configuration data persistently. This information is represented in Configuration objects. The actual configuration data is a Dictionary of properties inside a Configuration object.

There are two principally different ways to manage configurations. First there is the concept of a Managed Service, where configuration data is uniquely associated with an object registered with the service registry.

Next, there is the concept of a factory where the Configuration Admin service will maintain 0 or more Configuration objects for a Managed Service Factory that is registered with the Framework.

The first concept is intended for configuration data about “things/services” whose existence is defined externally, e.g. a specific printer. Factories are intended for “things/services” that can be created any number of times, e.g. a configuration for a DHCP server for different networks.

Bundles that require configuration should register a Managed Service or a Managed Service Factory in the service registry. A registration property named service.pid (persistent identifier or PID) must be used to identify this Managed Service or Managed Service Factory to the ConfigurationAdmin service.

When the ConfigurationAdmin detects the registration of a Managed Service, it checks its persistent storage for a configuration object whose service.pid property matches the PID service property
(service.pid) of the Managed Service. If found, it calls ManagedService.updated(Dictionary) method with the new properties. The implementation of a Configuration Admin service must run these callbacks asynchronously to allow proper synchronization.

When the Configuration Admin service detects a Managed Service Factory registration, it checks its storage for configuration objects whose service.factoryPid property matches the PID service property of the Managed Service Factory. For each such Configuration objects, it calls the ManagedServiceFactory.updated method asynchronously with the new properties. The calls to the updated method of a ManagedServiceFactory must be executed sequentially and not overlap in time.

In general, bundles having permission to use the Configuration Admin service can only access and modify their own configuration information. Accessing or modifying the configuration of other bundles requires ConfigurationPermission[location,CONFIGURE], where location is the configuration location.

Configuration objects can be bound to a specified bundle location or to a region (configuration location starts with ?). If a location is not set, it will be learned the first time a target is registered. If the location is learned this way, the Configuration Admin service must detect if the bundle corresponding to the location is uninstalled. If this occurs, the Configuration object must be unbound, that is its location field is set back to null.

If target's bundle location matches the configuration location it is always updated.

If the configuration location starts with ?, that is, the location is a region, then the configuration must be delivered to all targets registered with the given PID. If security is on, the target bundle must have ConfigurationPermission[location,TARGET], where location matches given the configuration location with wildcards as in the Filter substring match. The security must be verified using the org.osgi.framework.Bundle.hasPermission(Object) method on the target bundle.

If a target cannot be updated because the location does not match or it has no permission and security is active then the Configuration Admin service must not do the normal callback.

The method descriptions of this class refer to a concept of “the calling bundle”. This is a loose way of referring to the bundle which obtained the Configuration Admin service from the service registry. Implementations of ConfigurationAdmin must use a org.osgi.framework.ServiceFactory to support this concept.

**Concurrency** Thread-safe

**Provider Type** Consumers of this API must not implement this type

### 104.12.4.1 public static final String SERVICE_BUNDLELOCATION = "service.bundleLocation"

Configuration property naming the location of the bundle that is associated with a Configuration object. This property can be searched for but must not appear in the configuration dictionary for security reason. The property's value is of type String.

**Since** 1.1

### 104.12.4.2 public static final String SERVICE_FACTORYPID = "service.factoryPid"

Configuration property naming the Factory PID in the configuration dictionary. The property's value is of type String.

**Since** 1.1

### 104.12.4.3 public Configuration createFactoryConfiguration(String factoryPid) throws IOException

**factoryPid** PID of factory (not null).

- Create a new factory Configuration object with a new PID. The properties of the new Configuration object are null until the first time that its Configuration.update(Dictionary) method is called.
It is not required that the factoryPid maps to a registered Managed Service Factory.

The Configuration object is bound to the location of the calling bundle. It is possible that the same factoryPid has associated configurations that are bound to different bundles. Bundles should only see the factory configurations that they are bound to or have the proper permission.

Returns  A new Configuration object.

Throws  IOException – if access to persistent storage fails.

104.12.4.4  public Configuration createFactoryConfiguration(String factoryPid, String location) throws IOException

factoryPid  PID of factory (not null).

location  A bundle location string, or null.

□ Create a new factory Configuration object with a new PID. The properties of the new Configuration object are null until the first time that its Configuration.update(Dictionary) method is called.

It is not required that the factoryPid maps to a registered Managed Service Factory.

The Configuration is bound to the location specified. If this location is null it will be bound to the location of the first bundle that registers a Managed Service Factory with a corresponding PID. It is possible that the same factoryPid has associated configurations that are bound to different bundles. Bundles should only see the factory configurations that they are bound to or have the proper permission.

If the location starts with ? then the configuration must be delivered to all targets with the corresponding PID.

Returns  a new Configuration object.

Throws  IOException – if access to persistent storage fails.

SecurityException – when the require permissions are not available

Security  ConfigurationPermission[location,CONFIGURE] – if location is not null

ConfigurationPermission[",*,CONFIGURE"] – if location is null

104.12.4.5  public Configuration getConfiguration(String pid, String location) throws IOException

pid  Persistent identifier.

location  The bundle location string, or null.

□ Get an existing Configuration object from the persistent store, or create a new Configuration object.

If a Configuration with this PID already exists in Configuration Admin service return it. The location parameter is ignored in this case though it is still used for a security check.

Else, return a new Configuration object. This new object is bound to the location and the properties are set to null. If the location parameter is null, it will be set when a Managed Service with the corresponding PID is registered for the first time. If the location starts with ? then the configuration is bound to all targets that are registered with the corresponding PID.

Returns  An existing or new Configuration object.

Throws  IOException – if access to persistent storage fails.

SecurityException – when the require permissions are not available

Security  ConfigurationPermission[",*,CONFIGURE"] – if location is null or if the returned configuration c already exists and c.location is null

ConfigurationPermission[location,CONFIGURE] – if location is not null

ConfigurationPermission[c.location,CONFIGURE] – if the returned configuration c already exists and c.location is not null
104.12.6  public Configuration getConfiguration(String pid) throws IOException

    pid  persistent identifier.

    □  Get an existing or new Configuration object from the persistent store. If the Configuration object
    for this PID does not exist, create a new Configuration object for that PID, where properties are null.
    Bind its location to the calling bundle's location.

    Otherwise, if the location of the existing Configuration object is null, set it to the calling bundle's lo-
    cation.

    Returns  an existing or new Configuration matching the PID.

    Throws  IOException – if access to persistent storage fails.
    SecurityException – when the required permission is not available

104.12.7  public Configuration[] listConfigurations(String filter) throws IOException, InvalidSyntaxException

    filter  A filter string, or null to retrieve all Configuration objects.

    □  List the current Configuration objects which match the filter.

    Only Configuration objects with non-null properties are considered current. That is,
    Configuration.getProperties() is guaranteed not to return null for each of the returned Configura-
    tion objects.

    When there is no security on then all configurations can be returned. If security is on, the caller
    must have ConfigurationPermission[location,CONFIGURE].

    The syntax of the filter string is as defined in the Filter class. The filter can test any configuration
    properties including the following:
    •  service.pid - the persistent identity
    •  service.factoryPid - the factory PID, if applicable
    •  service.bundleLocation - the bundle location

    The filter can also be null, meaning that all Configuration objects should be returned.

    Returns  All matching Configuration objects, or null if there aren't any.

    Throws  IOException – if access to persistent storage fails
    InvalidSyntaxException – if the filter string is invalid
    Security  ConfigurationPermission[cache.location,CONFIGURE] – Only configurations c are returned for which
    the caller has this permission

104.12.5  public class ConfigurationEvent

    A Configuration Event.

    ConfigurationEvent objects are delivered to all registered ConfigurationListener service objects.
    ConfigurationEvents must be delivered in chronological order with respect to each listener.

    A type code is used to identify the type of event. The following event types are defined:

    •  CM_UPDATED
    •  CM_DELETED
    •  CM_LOCATION_CHANGED

    Additional event types may be defined in the future.
Security Considerations. ConfigurationEvent objects do not provide Configuration objects, so no sensitive configuration information is available from the event. If the listener wants to locate the Configuration object for the specified pid, it must use ConfigurationAdmin.

See Also: ConfigurationListener

Since: 1.2

Concurrency: Immutable

104.12.5.1 public static final int CM_DELETED = 2

A Configuration has been deleted.

This ConfigurationEvent type that indicates that a Configuration object has been deleted. An event is fired when a call to Configuration.delete() successfully deletes a configuration.

Since: 1.4

104.12.5.2 public static final int CM_LOCATION_CHANGED = 3

The location of a Configuration has been changed.

This ConfigurationEvent type that indicates that the location of a Configuration object has been changed. An event is fired when a call to Configuration.setBundleLocation(String) successfully changes the location.

Since: 1.4

104.12.5.3 public static final int CM_UPDATED = 1

A Configuration has been updated.

This ConfigurationEvent type that indicates that a Configuration object has been updated with new properties. An event is fired when a call to Configuration.update(Dictionary) successfully changes a configuration.

104.12.5.4 public ConfigurationEvent(ServiceReference<ConfigurationAdmin> reference, int type, String factoryPid, String pid)

reference The ServiceReference object of the Configuration Admin service that created this event.

type The event type. See getType().

factoryPid The factory pid of the associated configuration if the target of the configuration is a ManagedServiceFactory. Otherwise null if the target of the configuration is a ManagedService.

pid The pid of the associated configuration.

□ Constructs a ConfigurationEvent object from the given ServiceReference object, event type, and pids.

104.12.5.5 public String getFactoryPid()

□ Returns the factory pid of the associated configuration.

Returns: Returns the factory pid of the associated configuration if the target of the configuration is a ManagedServiceFactory. Otherwise null if the target of the configuration is a ManagedService.

104.12.5.6 public String getPid()

□ Returns the pid of the associated configuration.

Returns: Returns the pid of the associated configuration.

104.12.5.7 public ServiceReference<ConfigurationAdmin> getReference()

□ Return the ServiceReference object of the Configuration Admin service that created this event.

Returns: The ServiceReference object for the Configuration Admin service that created this event.
public int getType()

Return the type of this event.

The type values are:

- CM_UPDATED
- CM_DELETED
- CM_LOCATION_CHANGED

Returns The type of this event.

public classConfigurationException extends Exception

An Exception class to inform the Configuration Admin service of problems with configuration data.

public ConfigurationException(String property,String reason)

property name of the property that caused the problem, null if no specific property was the cause
reason reason for failure

Create aConfigurationException object.

public ConfigurationException(String property,String reason,Throwable cause)

property name of the property that caused the problem, null if no specific property was the cause
reason reason for failure
cause The cause of this exception.

Create aConfigurationException object.

Since 1.2

public Throwable getCause()

Returns the cause of this exception or null if no cause was set.

Returns The cause of this exception or null if no cause was set.

Since 1.2

public String getProperty()

Return the property name that caused the failure or null.

Returns name of property or null if no specific property caused the problem

public String getReason()

Return the reason for this exception.

Returns reason of the failure

public Throwable initCause(Throwable cause)

cause The cause of this exception.

Initializes the cause of this exception to the specified value.

Returns This exception.

Throws IllegalArgumentException– If the specified cause is this exception.
IllegalStateException– If the cause of this exception has already been set.
104.12.7 public interface ConfigurationListener

Listener for Configuration Events. When a ConfigurationEvent is fired, it is asynchronously delivered to all ConfigurationListeners.

ConfigurationListener objects are registered with the Framework service registry and are notified with a ConfigurationEvent object when an event is fired.

ConfigurationListener objects can inspect the received ConfigurationEvent object to determine its type, the pid of the Configuration object with which it is associated, and the Configuration Admin service that fired the event.

Security Considerations. Bundles wishing to monitor configuration events will require ServicePermission[ConfigurationListener,REGISTER] to register a ConfigurationListener service.

Since 1.2
Concurrency Thread-safe

104.12.7.1 public void configurationEvent(ConfigurationEvent event)

event The ConfigurationEvent.

Determines the equality of two ConfigurationPermission objects.

Returns true if obj is equivalent to this ConfigurationPermission; false otherwise.
104.12.8.5  
**public String getActions()**
- Returns the canonical string representation of the ConfigurationPermission actions.
- Always returns present ConfigurationPermission actions in the following order: “configure”, “target”

*Returns*  
Canonical string representation of the ConfigurationPermission actions.

104.12.8.6  
**public int hashCode()**
- Returns the hash code value for this object.

*Returns*  
Hash code value for this object.

104.12.8.7  
**public boolean implies(Permission p)**

- *p*  
  The target permission to check.
- Determines if a ConfigurationPermission object “implies” the specified permission.

*Returns*  
true if the specified permission is implied by this object; false otherwise.

104.12.8.8  
**public PermissionCollection newPermissionCollection()**
- Returns a new PermissionCollection object suitable for storing ConfigurationPermissions.

*Returns*  
A new PermissionCollection object.

104.12.9  
**public interface ConfigurationPlugin**

A service interface for processing configuration dictionary before the update.

A bundle registers a ConfigurationPlugin object in order to process configuration updates before they reach the Managed Service or Managed Service Factory. The Configuration Admin service will detect registrations of Configuration Plugin services and must call these services every time before it calls the ManagedService or ManagedServiceFactory updated method. The Configuration Plugin service thus has the opportunity to view and modify the properties before they are passed to the Managed Service or Managed Service Factory.

Configuration Plugin (plugin) services have full read/write access to all configuration information that passes through them.

The Integer service.cmRanking registration property may be specified. Not specifying this registration property, or setting it to something other than an Integer, is the same as setting it to the Integer zero. The service.cmRanking property determines the order in which plugins are invoked. Lower ranked plugins are called before higher ranked ones. In the event of more than one plugin having the same value of service.cmRanking, then the Configuration Admin service arbitrarily chooses the order in which they are called.

By convention, plugins with service.cmRanking < 0 or service.cmRanking > 1000 should not make modifications to the properties.

The Configuration Admin service has the right to hide properties from plugins, or to ignore some or all the changes that they make. This might be done for security reasons. Any such behavior is entirely implementation defined.

A plugin may optionally specify a cm.target registration property whose value is the PID of the Managed Service or Managed Service Factory whose configuration updates the plugin is intended to intercept. The plugin will then only be called with configuration updates that are targeted at the Managed Service or Managed Service Factory with the specified PID. Omitting the cm.target registration property means that the plugin is called for all configuration updates.

*Concurrency*  
Thread-safe
104.12.9.1  

**public static final String CM_RANKING = "service.cmRanking"**

A service property to specify the order in which plugins are invoked. This property contains an Integer ranking of the plugin. Not specifying this registration property, or setting it to something other than an Integer, is the same as setting it to the Integer zero. This property determines the order in which plugins are invoked. Lower ranked plugins are called before higher ranked ones.

*Since* 1.2

104.12.9.2  

**public static final String CM_TARGET = "cm.target"**

A service property to limit the Managed Service or Managed Service Factory configuration dictionaries a Configuration Plugin service receives. This property contains a String[] of PIDs. A Configuration Admin service must call a Configuration Plugin service only when this property is not set, or the target service’s PID is listed in this property.

104.12.9.3  

**public void modifyConfiguration(ServiceReference<?> reference, Dictionary<String, Object> properties)**

- **reference** reference to the Managed Service or Managed Service Factory
- **properties** The configuration properties. This argument must not contain the "service.bundleLocation" property. The value of this property may be obtained from the Configuration.getBundleLocation method.

View and possibly modify the a set of configuration properties before they are sent to the Managed Service or the Managed Service Factory. The Configuration Plugin services are called in increasing order of their service.cmRanking property. If this property is undefined or is a non-Integer type, 0 is used.

This method should not modify the properties unless the service.cmRanking of this plugin is in the range $0 \leq \text{service.cmRanking} \leq 1000$. If this method throws any Exception, the Configuration Admin service must catch it and should log it.

A Configuration Plugin will only be called for properties from configurations that have a location for which the Configuration Plugin has permission when security is active. When security is not active, no filtering is done.

104.12.10  

**public interface ManagedService**

A service that can receive configuration data from a Configuration Admin service.

A Managed Service is a service that needs configuration data. Such an object should be registered with the Framework registry with the service.pid property set to some unique identifier called a PID.

If the Configuration Admin service has a Configuration object corresponding to this PID, it will callback the updated() method of the ManagedService object, passing the properties of that Configuration object.

If it has no such Configuration object, then it calls back with a null properties argument. Registering a Managed Service will always result in a callback to the updated() method provided the Configuration Admin service is, or becomes active. This callback must always be done asynchronously.

Else, every time that either of the updated() methods is called on that Configuration object, the ManagedService.updated() method with the new properties is called. If the delete() method is called on that Configuration object, ManagedService.updated() is called with a null for the properties parameter. All these callbacks must be done asynchronously.

The following example shows the code of a serial port that will create a port depending on configuration information.

```java
class SerialPort implements ManagedService {
```
synchronized void open(CommPortIdentifier id, BundleContext context) {
    this.id = id;
    registration = context.registerService(
        ManagedService.class.getName(),
        this,
        getDefaults()
    );
}

Hashtable getDefaults() {
    Hashtable defaults = new Hashtable();
    defaults.put("port", id.getName());
    defaults.put("product", "unknown");
    defaults.put("baud", "9600");
    defaults.put( Constants.SERVICE_PID,
        "com.acme.serialport." + id.getName() );
    return defaults;
}

public synchronized void updated( Dictionary configuration ) {
    if ( configuration == null )
        registration.setProperties( getDefaults() );
    else {
        setSpeed( configuration.get("baud") );
        registration.setProperties( configuration );
    }
}

As a convention, it is recommended that when a Managed Service is updated, it should copy all the properties it does not recognize into the service registration properties. This will allow the Configuration Admin service to set properties on services which can then be used by other applications.

Normally, a single Managed Service for a given PID is given the configuration dictionary, this is the configuration that is bound to the location of the registering bundle. However, when security is on, a Managed Service can have Configuration Permission to also be updated for other locations.

Concurrency Thread-safe

104.12.10.1 public void updated(Dictionary<String,?> properties) throws ConfigurationException

properties A copy of the Configuration properties, or null. This argument must not contain the "service.bundleLocation" property. The value of this property may be obtained from the Configuration.getBundleLocation method.

Update the configuration for a Managed Service.

When the implementation of updated(Dictionary) detects any kind of error in the configuration properties, it should create a new ConfigurationException which describes the problem. This can allow a management system to provide useful information to a human administrator.
If this method throws any other Exception, the Configuration Admin service must catch it and should log it.

The Configuration Admin service must call this method asynchronously with the method that initiated the callback. This implies that implementors of Managed Service can be assured that the callback will not take place during registration when they execute the registration in a synchronized method.

If the the location allows multiple managed services to be called back for a single configuration then the callbacks must occur in service ranking order. Changes in the location must be reflected by deleting the configuration if the configuration is no longer visible and updating when it becomes visible.

If no configuration exists for the corresponding PID, or the bundle has no access to the configuration, then the bundle must be called back with a null to signal that CM is active but there is no data.

Throws  ConfigurationException– when the update fails

Security  ConfigurationPermission[c.location,TARGET] – Required by the bundle that registered this service

104.12.11  public interface ManagedServiceFactory

Manage multiple service instances. Bundles registering this interface are giving the Configuration Admin service the ability to create and configure a number of instances of a service that the implementing bundle can provide. For example, a bundle implementing a DHCP server could be instantiated multiple times for different interfaces using a factory.

Each of these service instances is represented, in the persistent storage of the Configuration Admin service, by a factory Configuration object that has a PID. When such a Configuration is updated, the Configuration Admin service calls the ManagedServiceFactory updated method with the new properties. When updated is called with a new PID, the Managed Service Factory should create a new factory instance based on these configuration properties. When called with a PID that it has seen before, it should update that existing service instance with the new configuration information.

In general it is expected that the implementation of this interface will maintain a data structure that maps PIDs to the factory instances that it has created. The semantics of a factory instance are defined by the Managed Service Factory. However, if the factory instance is registered as a service object with the service registry, its PID should match the PID of the corresponding Configuration object (but it should not be registered as a Managed Service!).

An example that demonstrates the use of a factory. It will create serial ports under command of the Configuration Admin service.

class SerialPortFactory
    implements ManagedServiceFactory {
        ServiceRegistration registration;
        Hashtable ports;
        void start(BundleContext context) {
            Hashtable properties = new Hashtable();
            properties.put( Constants.SERVICE_PID,
                "com.acme.serialportfactory" );
            registration = context.registerService(ManagedServiceFactory.class.getName(),
                this,
                properties
            );
        }

        public void updated( String pid,
            Dictionary properties  ) {
String portName = (String) properties.get("port");
SerialPortService port =
    (SerialPort) ports.get( pid );
if ( port == null ) {
    port = new SerialPortService();
    ports.put( pid, port );
    port.open();
}
if ( port.getPortName().equals(portName) )
    return;
port.setPortName( portName );
}

public void deleted( String pid ) {
    SerialPortService port =
        (SerialPort) ports.get( pid );
    port.close();
    ports.remove( pid );
}
...

Concurrency Thread-safe

104.12.11.1 public void deleted(String pid)
pid  the PID of the service to be removed

- Remove a factory instance. Remove the factory instance associated with the PID. If the instance was registered with the service registry, it should be unregistered. The Configuration Admin must call deleted for each instance it received in updated(String, Dictionary).

If this method throws any Exception, the Configuration Admin service must catch it and should log it.

The Configuration Admin service must call this method asynchronously.

104.12.11.2 public String getName()

- Return a descriptive name of this factory.

Returns  the name for the factory, which might be localized

104.12.11.3 public void updated(String pid,Dictionary<String,?> properties) throws ConfigurationException

- The PID for this configuration.

properties  A copy of the configuration properties. This argument must not contain the service.bundleLocation property. The value of this property may be obtained from the Configuration.getBundleLocation method.

- Create a new instance, or update the configuration of an existing instance. If the PID of the Configuration object is new for the Managed Service Factory, then create a new factory instance, using the configuration properties provided. Else, update the service instance with the provided properties.

If the factory instance is registered with the Framework, then the configuration properties should be copied to its registry properties. This is not mandatory and security sensitive properties should obviously not be copied.

If this method throws any Exception, the Configuration Admin service must catch it and should log it.

When the implementation of updated detects any kind of error in the configuration properties, it should create a new ConfigurationException which describes the problem.
The Configuration Admin service must call this method asynchronously. This implies that implementors of the ManagedServiceFactory class can be assured that the callback will not take place during registration when they execute the registration in a synchronized method.

If the security allows multiple managed service factories to be called back for a single configuration then the callbacks must occur in service ranking order.

It is valid to create multiple factory instances that are bound to different locations. Managed Service Factory services must only be updated with configurations that are bound to their location or that start with the ? prefix and for which they have permission. Changes in the location must be reflected by deleting the corresponding configuration if the configuration is no longer visible or updating when it becomes visible.

*Throws* ConfigurationException—when the configuration properties are invalid.

*Security* ConfigurationPermission[c.location,TARGET]—Required by the bundle that registered this service

### 104.12.12

**public interface SynchronousConfigurationListener extends ConfigurationListener**

Synchronous Listener for Configuration Events. When a ConfigurationEvent is fired, it is synchronously delivered to all SynchronousConfigurationListeners.

SynchronousConfigurationListener objects are registered with the Framework service registry and are synchronously notified with a ConfigurationEvent object when an event is fired.

SynchronousConfigurationListener objects can inspect the received ConfigurationEvent object to determine its type, the PID of the Configuration object with which it is associated, and the Configuration Admin service that fired the event.

Security Considerations. Bundles wishing to synchronously monitor configuration events will require ServicePermission[SynchronousConfigurationListener,REGISTER] to register a SynchronousConfigurationListener service.

*Since* 1.5

*Concurrency* Thread-safe

### 104.13

**Changes**

- Clarified that collection property values may have an ordering that must be preserved.
Metatype Service Specification

Version 1.2

Introduction

The Metatype specification defines interfaces that allow bundle developers to describe attribute types in a computer readable form using so-called metadata.

The purpose of this specification is to allow services to specify the type information of data that they can use as arguments. The data is based on attributes, which are key/value pairs like properties.

A designer in a type-safe language like Java is often confronted with the choice of using the language constructs to exchange data or using a technique based on attributes/properties that are based on key/value pairs. Attributes provide an escape from the rigid type safety requirements of modern programming languages.

Type-safety works very well for software development environments in which multiple programmers work together on large applications or systems, but often lacks the flexibility needed to receive structured data from the outside world.

The attribute paradigm has several characteristics that make this approach suitable when data needs to be communicated between different entities which "speak" different languages. Attributes are uncomplicated, resilient to change, and allow the receiver to dynamically adapt to different types of data.

As an example, the OSGi framework Specifications define several attribute types which are used in a Framework implementation, but which are also used and referenced by other OSGi specifications such as the Configuration Admin Service Specification on page 63. A Configuration Admin service implementation deploys attributes (key/value pairs) as configuration properties.

The Meta Type Service provides a unified access point to the Meta Type information that is associated with bundles. This Meta Type information can be defined by an XML resource in a bundle (OSGI-INF/metatype directories must be scanned for any XML resources), it can come from the Meta Type Provider service, or it can be obtained from Managed Service or Managed Service Factory services.

105.1.1 Essentials

- **Conceptual model** - The specification must have a conceptual model for how classes and attributes are organized.
- **Standards** - The specification should be aligned with appropriate standards, and explained in situations where the specification is not aligned with, or cannot be mapped to, standards.
- **Remote Management** - Remote management should be taken into account.
- **Size** - Minimal overhead in size for a bundle using this specification is required.
- **Localization** - It must be possible to use this specification with different languages at the same time. This ability allows servlets to serve information in the language selected in the browser.
- **Type information** - The definition of an attribute should contain the name (if it is required), the cardinality, a label, a description, labels for enumerated values, and the Java class that should be used for the values.
- **Validation** - It should be possible to validate the values of the attributes.
105.1.2 **Entities**

- **Meta Type Service** - A service that provides a unified access point for meta type information.
- **Attribute** - A key/value pair.
- **PID** - A unique persistent ID, defined in configuration management.
- **Attribute Definition** - Defines a description, name, help text, and type information of an attribute.
- **Object Class Definition** - Defines the type of a datum. It contains a description and name of the type plus a set of AttributeDefinition objects.
- **Meta Type Provider** - Provides access to the object classes that are available for this object. Access uses the PID and a locale to find the best ObjectClassDefinition object.
- **Meta Type Information** - Provides meta type information for a bundle.

![Class Diagram Meta Type Service, org.osgi.service.metatype](image)

105.1.3 **Operation**

The Meta Type service defines a rich dynamic typing system for properties. The purpose of the type system is to allow reasonable User Interfaces to be constructed dynamically.

The type information is normally carried by the bundles themselves. Either by implementing the MetaTypeProvider interface on the Managed Service or Managed Service Factory, by carrying one or more XML resources that define a number of Meta Types in the OSGI-INF/metatype directories, or registering a Meta Type Provider as a service. Additionally, a Meta Type service could have other sources that are not defined in this specification.

The Meta Type Service provides unified access to Meta Types that are carried by the resident bundles. The Meta Type Service collects this information from the bundles and provides uniform access to it. A client can requests the Meta Type Information associated with a particular bundle. The MetaTypeInformation object provides a list of ObjectClassDefinition objects for a bundle. These objects define all the information for a specific object class. An object class is some descriptive information and a set of named attributes (which are key/value pairs).

Access to Object Class Definitions is qualified by a locale and a Persistent IDentity (PID). This specification does not specify what the PID means. One application is OSGi Configuration Management where a PID is used by the Managed Service and Managed Service Factory services. In general, a PID should be regarded as the name of a variable where an Object Class Definition defines its type.

105.2 **Attributes Model**

The Framework uses the LDAP filter syntax for searching the Framework registry. The usage of the attributes in this specification and the Framework specification closely resemble the LDAP attribute...
model. Therefore, the names used in this specification have been aligned with LDAP. Consequently, the interfaces which are defined by this Specification are:

- AttributeDefinition
- ObjectClassDefinition
- MetaTypeProvider

These names correspond to the LDAP attribute model. For further information on ASN.1-defined attributes and X.500 object classes and attributes, see [2] Understanding and Deploying LDAP Directory services.

The LDAP attribute model assumes a global name-space for attributes, and object classes consist of a number of attributes. So, if an object class inherits the same attribute from different parents, only one copy of the attribute must become part of the object class definition. This name-space implies that a given attribute, for example cn, should always be the common name and the type must always be a String. An attribute cn cannot be an Integer in another object class definition. In this respect, the OSGi approach towards attribute definitions is comparable with the LDAP attribute model.

105.3 Object Class Definition

The ObjectClassDefinition interface is used to group the attributes which are defined in AttributeDefinition objects.

An ObjectClassDefinition object contains the information about the overall set of attributes and has the following elements:

- A name which can be returned in different locales.
- A global name-space in the registry, which is the same condition as LDAP/X.500 object classes. In these standards the OSI Object Identifier (OID) is used to uniquely identify object classes. If such an OID exists, (which can be requested at several standard organizations, and many companies already have a node in the tree) it can be returned here. Otherwise, a unique id should be returned. This id can be a Java class name (reverse domain name) or can be generated with a GUID algorithm. All LDAP-defined object classes already have an associated OID. It is strongly advised to define the object classes from existing LDAP schemes which provide many preexisting OIDs. Many such schemes exist ranging from postal addresses to DHCP parameters.
- A human-readable description of the class.
- A list of attribute definitions which can be filtered as required, or optional. Note that in X.500 the mandatory or required status of an attribute is part of the object class definition and not of the attribute definition.
- An icon, in different sizes.

105.4 Attribute Definition

The AttributeDefinition interface provides the means to describe the data type of attributes.

The AttributeDefinition interface defines the following elements:

- Defined names (final ints) for the data types as restricted in the Framework for the attributes, called the syntax in OSI terms, which can be obtained with the getType() method.
- AttributeDefinition objects should use an ID that is similar to the OID as described in the ID field for ObjectClassDefinition.
- A localized name intended to be used in user interfaces.
- A localized description that defines the semantics of the attribute and possible constraints, which should be usable for tooltips.
- An indication if this attribute should be stored as a unique value, a Vector, or an array of values, as well as the maximum cardinality of the type.
- The data type, as limited by the Framework service registry attribute types.
- A validation function to verify if a possible value is correct.
- A list of values and a list of localized labels. Intended for popup menus in GUIs, allowing the user to choose from a set.
- A default value (String[]). The return depends on the following cases:
  - not specified - Return null if this attribute is not specified.
  - cardinality = 0 - Return an array with one element.
  - otherwise - Return an array with less or equal than the absolute value of cardinality, possibly empty if the value is an empty string.

105.5 Meta Type Service

The Meta Type Service provides unified access to Meta Type information that is associated with a Bundle. It can get this information through the following means:

- **Meta Type Resource** - A bundle can provide one or more XML resources that are contained in its JAR file. These resources contain an XML definition of meta types as well as to what PIDs these Meta Types apply. These XML resources must reside in the OSGI-INF/metatype directories of the bundle (including any fragments).
- **Managed Service [Factory] objects** - As defined in the configuration management specification, ManagedService and ManagedServiceFactory service objects can optionally implement the MetaTypeProvider interface. The Meta Type Service will only search for MetaTypeProvider objects if no meta type resources are found in the bundle.
- **Meta Type Provider service** - Bundles can register Meta Type Provider services to dynamically provide meta types for PIDs and factory PIDs.

*Figure 105.2 Sources for Meta Types*

This model is depicted in Figure 105.2.

The Meta Type Service can therefore be used to retrieve meta type information for bundles which contain Meta Type resources or which provide their own MetaTypeProvider objects. The MetaTypeService interface has a single method:

- **getMetaTypeInformation(Bundle)** - Given a bundle, it must return the Meta Type Information for that bundle, even if there is no meta type information available at the moment of the call.
The returned `MetaTypeInformation` object maintains a map of PID to `ObjectClassDefinition` objects. The map is keyed by locale and PID. The list of maintained PIDs is available from the `MetaTypeInformation` object with the following methods:

- `getPids()` - PIDs for which Meta Types are available.
- `getFactoryPids()` - PIDs associated with Managed Service Factory services.

These methods and their interaction with the Meta Type resource are described in *Use of the Designate Element* on page 116.

The `MetaTypeInformation` interface extends the `MetaTypeProvider` interface. The `MetaTypeProvider` interface is used to access meta type information. It supports locale dependent information so that the text used in `AttributeDefinition` and `ObjectClassDefinition` objects can be adapted to different locales.

Which locales are supported by the `MetaTypeProvider` object are defined by the implementer or the meta type resources. The list of available locales can be obtained from the `MetaTypeProvider` object.

The `MetaTypeProvider` interface provides the following methods:

- `getObjectClassDefinition(String,String)` - Get access to an `ObjectClassDefinition` object for the given PID. The second parameter defines the locale.
- `getLocales()` - List the locales that are available.

Locale objects are represented in `String` objects because not all profiles support Locale. The `String` holds the standard Locale presentation of:

```
locale = language ( '__' country ( '__' variation))
language ::= < defined by ISO 3166 >
country ::= < defined by ISO 639 >
```

For example, en, nl_BE, en_CA_posix are valid locales. The use of null for locale indicates that `java.util.Locale.getDefault()` must be used.

The Meta Type Service implementation class is the main class. It registers the `org.osgi.service.metatype.MetaTypeService` service and has a method to get a `MetaTypeInformation` object for a bundle.

Following is some sample code demonstrating how to print out all the Object Class Definitions and Attribute Definitions contained in a bundle:

```java
void printMetaTypes( MetaTypeService mts,Bundle b ) {
    MetaTypeInformation mti =
        mts.getMetaTypeInformation(b);
    String [] pids = mti.getPids();
    String [] locales = mti.getLocales();
    for ( int locale = 0; locale<locales.length; locale++) {
        System.out.println("Locale "+locales[locale]);
        for (int i=0; i<pids.length; i++) {
            ObjectClassDefinition ocd =
                mti.getObjectClassDefinition(pids[i], null);
            AttributeDefinition[] ads =
                ocd.getAttributeDefinitions(
                    ObjectClassDefinition.ALL);
            for (int j=0; j<ads.length; j++) {
                System.out.println("OCD="+ocd.getName() + "AD="+ads[j].getName());
            }
        }
    }
}
```
105.6 Meta Type Provider Service

A Meta Type Provider service allows third party contributions to the internal Object Class Definition repository. A Meta Type Provider can contribute multiple PIDs, both factory and singleton PIDs. A Meta Type Provider service must register with both or one of the following service properties:

- **METATYPE_PID** - (String+) Provides a list of PIDs that this Meta Type Provider can provide Object Class Definitions for. The listed PIDs are intended to be used as normal singleton PIDs used by Managed Services.
- **METATYPE_FACTORY_PID** - (String+) Provides a list of factory PIDs that this Meta Type Provider can provide Object Class Definitions for. The listed PIDs are intended to be used as factory PIDs used by Managed Service Factories.

The Object Class Definitions must originate from the bundle that registered the Meta Type Provider service. Third party extenders should therefore use the bundle of their extender. A Meta Type Service must report these Object Class Definitions on the Meta Type Information of the registering bundle, merged with any other information from that bundle.

The Meta Type Service must track these Meta Type Provider services and make their Meta Types available as if they were provided on the Managed Service (Factory) services. The Meta Types must become unavailable when the Meta Type Provider service is unregistered.

105.7 Using the Meta Type Resources

A bundle that wants to provide meta type resources must place these resources in the OSGI-INF/metatype directory. The name of the resource must be a valid bundle entry path. All resources in that directory must be meta type documents. Fragments can contain additional meta type resources in the same directory and they must be taken into account when the meta type resources are searched. A meta type resource must be encoded in UTF-8.

The MetaType Service must support localization of the

- name
- icon
- description
- label attributes

The localization mechanism must be identical using the same mechanism as described in the Core module layer, see Localization, using the same property resource. However, it is possible to override the property resource in the meta type definition resources with the localization attribute of the MetaData element.

The Meta Type Service must examine the bundle and its fragments to locate all localization resources for the localization base name. From that list, the Meta Type Service derives the list of locales which are available for the meta type information. This list can then be returned by MetaTypeInformation.getLocales method. This list can change at any time because the bundle could be refreshed. Clients should be prepared that this list changes after they received it.
105.7.1 XML Schema of a Meta Type Resource

This section describes the schema of the meta type resource. This schema is not intended to be used during runtime for validating meta type resources. The schema is intended to be used by tools and external management systems.

The XML namespace for meta type documents must be:

http://www.osgi.org/xmlns/metatype/v1.2.0

The namespace abbreviation should be metatype. I.e. the following header should be:

```
<metatype:MetaData
    xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

The file can be found in the osgi.jar file that can be downloaded from the www.osgi.org web site.

Figure 105.3 XML Schema Instance Structure (Type name = Element name)

```
MetaData ::= OCD* Designate*

OCD ::= AD+ Icon
AD ::= Option*

Designate ::= Object
Object ::= Attribute *
```

The element structure of the XML file is:
Attribute ::= Value *

The different elements are described in Table 105.1.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Deflt</th>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaData</td>
<td></td>
<td></td>
<td></td>
<td>Top Element</td>
</tr>
<tr>
<td>localization</td>
<td></td>
<td>string</td>
<td></td>
<td>Points to the Properties file that can localize this XML. See Localization in OSGi Core Release 6.</td>
</tr>
<tr>
<td>OCD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>&lt;&gt;</td>
<td>string</td>
<td>getName()</td>
<td>A human readable name that can be localized.</td>
</tr>
<tr>
<td>description</td>
<td></td>
<td></td>
<td>getDescription()</td>
<td>A human readable description of the Object Class Definition that can be localized.</td>
</tr>
<tr>
<td>id</td>
<td>&lt;&gt;</td>
<td></td>
<td>getID()</td>
<td>A unique id, cannot be localized.</td>
</tr>
<tr>
<td>Designate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pid</td>
<td>&lt;&gt;</td>
<td>string</td>
<td></td>
<td>An association between one PID and an Object Class Definition. This element designates a PID to be of a certain type.</td>
</tr>
<tr>
<td>factoryPid</td>
<td></td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bundle</td>
<td></td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>relevant</td>
<td>optional</td>
<td>false</td>
<td>boolean</td>
<td></td>
</tr>
</tbody>
</table>

If the factoryPid attribute is set, this Designate element defines a factory configuration for the given factory. If it is not set or empty, it designates a singleton configuration. The PID can be a Targeted PID, if factoryPid is not set or empty. Either pid or factoryPid must be specified. See Use of the Designate Element on page 116.

The value is used to set the location of any configuration created using this Meta Type resource. This may contain a bundle location or a multi-location. In a Meta Type resource, using the wildcard value ("*" \u002A) indicates the bundle location of the bundle containing the resource must be used as the location. See Location Binding on page 69.

This is an optional attribute but can be mandatory in certain usage schemes, for example the Autoconf Resource Processor.

If true, then this Designate element is optional, errors during processing must be ignored.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Deflt</th>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>merge</td>
<td>false</td>
<td>boolean</td>
<td></td>
<td>If the PID refers to an existing variable, then merge the properties with the existing properties if this attribute is true. Otherwise, replace the properties.</td>
</tr>
</tbody>
</table>

**AD**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>getName()</td>
<td>A localizable name for the Attribute Definition.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>getDescription()</td>
<td>A localizable description for the Attribute Definition.</td>
</tr>
<tr>
<td>id</td>
<td></td>
<td>getID()</td>
<td>The unique ID of the Attribute Definition.</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td>getType()</td>
<td>The type of an attribute is an enumeration of the different scalar types. The string is mapped to one of the constants on the AttributeDefinition interface. Valid values, which are defined in the Scalar type, are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>String ↔ STRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Long ↔ LONG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Double ↔ DOUBLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Float ↔ FLOAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Integer ↔ INTEGER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Char ↔ CHARACTER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Boolean ↔ BOOLEAN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Short ↔ SHORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Password ↔ PASSWORD</td>
</tr>
<tr>
<td>cardinality</td>
<td></td>
<td>getCardinality()</td>
<td>The number of elements an instance can take. Positive numbers describe an array ([])) and negative numbers describe a Vector object.</td>
</tr>
<tr>
<td>min</td>
<td>string</td>
<td>validate(String)</td>
<td>A validation value. This value is not directly available from the AttributeDefinition interface. However, the validate(String) method must verify this. The semantics of this field depend on the type of this Attribute Definition.</td>
</tr>
<tr>
<td>max</td>
<td>string</td>
<td>validate(String)</td>
<td>A validation value. Similar to the min field. When min or max are numbers, attribute values with a numeric data type are valid if min &lt;= value &lt;= max. Attribute values with a string (or equivalent) data type are valid if min &lt;= value.length() &lt;= max.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Deflt</td>
<td>Type</td>
<td>Method</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
<td>----------------------</td>
</tr>
<tr>
<td>default</td>
<td></td>
<td>string</td>
<td>getDefaultValue()</td>
</tr>
<tr>
<td>required</td>
<td>true</td>
<td>boolean</td>
<td></td>
</tr>
</tbody>
</table>

**Option**

| label     | <>   | string | getOptionLabels()    | The label                                                                   |
| value     | <>   | string | getOptionValues()    | The value                                                                   |
| Icon      |      |        |                      | An icon definition.                                                         |
| resource  | <>   | string | getIcon(int)         | The resource is a URL. The base URL is assumed to be the root of the bundle containing the XML file. That is, this URL can reference another resource in the bundle using a relative URL. The number of pixels of the icon, maps to the size parameter of the getIcon(int) method. |
| size      | <>   | string | getIcon(int)         | A definition of an instance. A reference to the id attribute of an OCD element. I.e. this attribute defines the OCD type of this object. A value for an attribute of an object. A reference to the id of the AD in the OCD as referenced by the parent Object. The content of the attributes. If this is an array, the content must be separated by commas (`,` \u002C). Commas must be escaped as described at the default attribute of the AD element. Holds a single value. This element can be repeated multiple times under an Attribute |

### 105.7.2 Use of the Designate Element

For the MetaType Service, the Designate definition is used to declare the available PIDs and factory PIDs; the Attribute elements are never used by the MetaType service.
The `getPids()` method returns an array of PIDs that were specified in the `pid` attribute of the `Object` elements. The `getFactoryPids()` method returns an array of the `factoryPid` attributes. For factories, the related `pid` attribute is ignored because all instances of a factory must share the same meta type.

The following example shows a metatype reference to a singleton configuration and a factory configuration.

```xml
<Designate pid="com.acme.designate.1">
    <Object ocdref="com.acme.designate"/>
</Designate>
<Designate factoryPid="com.acme.designate.factory"
    bundle="*">
    <Object ocdref="com.acme.designate"/>
</Designate>
```

Other schemes can embed the `Object` element in the `Designate` element to define actual instances for the Configuration Admin service. In that case the `pid` attribute must be used together with the `factoryPid` attribute. However, in that case an aliasing model is required because the Configuration Admin service does not allow the creator to choose the Configuration object's PID.

### Example Metadata File

This example defines a meta type file for a Person record, based on ISO attribute types. The ids that are used are derived from ISO attributes.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<MetaData
    xmlns="http://www.osgi.org/xmlns/metatype/v1.2.0"
    localization="person">
    <OCD name="%person" id="2.5.6.6"
        description="%Person Record">
        <AD name="%sex" id="2.5.4.12" type="Integer">
            <Option label="%male" value="1"/>
            <Option label="%Female" value="0"/>
        </AD>
        <AD name="%sn" id="2.5.4.4" type="String"/>
        <AD name="%cn" id="2.5.4.3" type="String"/>
        <AD name="%seeAlso" id="2.5.4.34" type="String"
            cardinality="8" default="http://www.google.com,
            http://www.yahoo.com"/>
        <AD name="%telNumber" id="2.5.4.20" type="String"/>
    </OCD>

    <Designate pid="com.acme.addressbook">
        <Object ocdref="2.5.6.6"/>
    </Designate>
</MetaData>
```

Translations for this file, as indicated by the localization attribute must be stored in the root directory (e.g. `person_du_NL.properties`). The default localization base name for the properties is `OSGI-INF/l10n/bundle`, but can be overridden by the manifest `Bundle-Localization` header and the localization attribute of the Meta Data element. The property files have the base name of `person`. The Dutch, French and English translations could look like:

```properties
person_du_NL.properties:
person=Persoon
```
Object

The OCD element can be used to describe the possible contents of a Dictionary object. In this case, the attribute name is the key. The Object element can be used to assign a value to a Dictionary object.

For example:

```xml
<Designate pid="com.acme.b">
  <Object ocdref="b">
    <Attribute adref="foo" content="Zaphod Beeblebrox"/>
    <Attribute adref="bar">
      <Value>1</Value>
      <Value>2</Value>
      <Value>3</Value>
      <Value>4</Value>
      <Value>5</Value>
    </Attribute>
  </Object>
</Designate>
```
105.9 XML Schema

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.2.0"
    targetNamespace="http://www.osgi.org/xmlns/metatype/v1.2.0"
    version="1.2.1">

    <element name="MetaData" type="metatype:Tmetadata" />

    <complexType name="Tmetadata">
        <sequence>
            <element name="OCD" type="metatype:Tocd" minOccurs="0"
                maxOccurs="unbounded" />
            <element name="Designate" type="metatype:Tdesignate"
                minOccurs="0" maxOccurs="unbounded" />
            <!-- It is non-deterministic, per W3C XML Schema 1.0: http://www.w3.org/TR/xmlschema-1/#cos-nonambig
            to use namespace="##any" below. -->
            <any namespace="##other" processContents="lax" minOccurs="0"
                maxOccurs="unbounded" />
        </sequence>
        <attribute name="localization" type="string" use="optional" />
        <anyAttribute processContents="lax" />
    </complexType>

    <complexType name="Tocd">
        <sequence>
            <element name="AD" type="metatype:Tad" minOccurs="1"
                maxOccurs="unbounded" />
            <element name="Icon" type="metatype:Ticon" minOccurs="0"
                maxOccurs="unbounded" />
            <!-- It is non-deterministic, per W3C XML Schema 1.0: http://www.w3.org/TR/xmlschema-1/#cos-nonambig
            to use namespace="##any" below. -->
            <any namespace="##other" processContents="lax" minOccurs="0"
                maxOccurs="unbounded" />
        </sequence>
        <attribute name="name" type="string" use="required" />
        <attribute name="description" type="string" use="optional" />
        <attribute name="id" type="string" use="required" />
        <attribute name="name" type="string" use="optional" />
        <anyAttribute processContents="lax" />
    </complexType>

    <complexType name="Tad">
        <sequence>
            <element name="Option" type="metatype:Toption" minOccurs="0"
                maxOccurs="unbounded" />
            <!-- It is non-deterministic, per W3C XML Schema 1.0: http://www.w3.org/TR/xmlschema-1/#cos-nonambig
            to use namespace="##any" below. -->
            <any namespace="##other" processContents="lax" minOccurs="0"
                maxOccurs="unbounded" />
        </sequence>
        <attribute name="name" type="string" use="optional" />
        <attribute name="description" type="string" use="optional" />
        <attribute name="id" type="string" use="required" />
        <attribute name="type" type="metatype:Tscalar" use="required" />
        <attribute name="cardinality" type="int" use="optional"
            default="0" />
        <attribute name="min" type="string" use="optional" />
        <attribute name="max" type="string" use="optional" />
        <attribute name="default" type="string" use="optional" />
        <attribute name="required" type="boolean" use="optional"
            default="true" />
        <anyAttribute processContents="lax" />
    </complexType>

    <complexType name="Tobject">
        <sequence>
            <element name="Attribute" type="metatype:Tattribute"
                minOccurs="0" maxOccurs="unbounded" />
            <!-- It is non-deterministic, per W3C XML Schema 1.0: http://www.w3.org/TR/xmlschema-1/#cos-nonambig
            to use namespace="##any" below. -->
            <any namespace="##other" processContents="lax" minOccurs="0"
                maxOccurs="unbounded" />
        </sequence>
    </complexType>

</schema>
```
<complexType name="Tattribute">
  <sequence>
    <element name="Value" type="string" minOccurs="0" maxOccurs="unbounded" />
    <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
  </sequence>
  <attribute name="adref" type="string" use="required" />
  <attribute name="content" type="string" use="optional" />
  <anyAttribute processContents="lax" />
</complexType>

<complexType name="Tdesignate">
  <sequence>
    <element name="Object" type="metatype:Tobject" minOccurs="1" maxOccurs="1" />
    <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
  </sequence>
  <attribute name="pid" type="string" use="optional" />
  <attribute name="factoryPid" type="string" use="optional" />
  <attribute name="bundle" type="string" use="optional" />
  <attribute name="optional" type="boolean" default="false" use="optional" />
  <attribute name="merge" type="boolean" default="false" use="optional" />
  <anyAttribute processContents="lax" />
</complexType>

<simpleType name="Tscalar">
  <restriction base="string">
    <enumeration value="String" />
    <enumeration value="Long" />
    <enumeration value="Double" />
    <enumeration value="Float" />
    <enumeration value="Integer" />
    <enumeration value="Byte" />
    <enumeration value="Char" />
    <enumeration value="Boolean" />
    <enumeration value="Short" />
    <enumeration value="Password" />
  </restriction>
</simpleType>

<complexType name="Toption">
  <sequence>
    <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
  </sequence>
  <attribute name="label" type="string" use="required" />
  <attribute name="value" type="string" use="required" />
  <anyAttribute processContents="lax" />
</complexType>

<complexType name="Ticon">
  <sequence>
    <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
  </sequence>
  <attribute name="resource" type="string" use="required" />
  <attribute name="size" type="positiveInteger" use="required" />
  <anyAttribute processContents="lax" />
</complexType>

<attribute name="must-understand" type="boolean">
  <annotation>
    <documentation xml:lang="en">
      This attribute should be used by extensions to documents to require that the document consumer understand the
    </documentation>
  </annotation>
</attribute>
105.10 Limitations

The OSGi MetaType specification is intended to be used for simple applications. It does not, therefore, support recursive data types, mixed types in arrays/vectors, or nested arrays/vectors.

105.11 Related Standards

One of the primary goals of this specification is to make metatype information available at runtime with minimal overhead. Many related standards are applicable to metatypes; except for Java beans, however, all other metatype standards are based on document formats (e.g. XML). In the OSGi framework, document format standards are deemed unsuitable due to the overhead required in the execution environment (they require a parser during run-time).

Another consideration is the applicability of these standards. Most of these standards were developed for management systems on platforms where resources are not necessarily a concern. In this case, a metatype standard is normally used to describe the data structures needed to control some other computer via a network. This other computer, however, does not require the metatype information as it is implementing this information.

In some traditional cases, a management system uses the metatype information to control objects in an OSGi framework. Therefore, the concepts and the syntax of the metatype information must be mappable to these popular standards. Clearly, then, these standards must be able to describe objects in an OSGi framework. This ability is usually not a problem, because the metatype languages used by current management systems are very powerful.

105.12 Security Considerations

Special security issues are not applicable for this specification.

105.13 org.osgi.service.metatype

Metatype Package Version 1.2.

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.metatype; version="[1.2,2.0)"

Example import for providers implementing the API in this package:
Import-Package: org.osgi.service.metatype; version="[1.2,1.3)"

105.13.1 Summary

- AttributeDefinition - An interface to describe an attribute.
• MetaTypeInformation - A MetaType Information object is created by the MetaTypeService to return meta type information for a specific bundle.
• MetaTypeProvider - Provides access to metatypes.
• MetaTypeService - The MetaType Service can be used to obtain meta type information for a bundle.
• ObjectClassDefinition - Description for the data type information of an objectclass.

105.13.2 public interface AttributeDefinition

An interface to describe an attribute. An AttributeDefinition object defines a description of the data type of a property/attribute.

Concurrency Thread-safe

105.13.2.1 public static final int BIGDECIMAL = 10

The BIGDECIMAL (10) type. Attributes of this type should be stored as BigDecimal, Vector with BigDecimal or BigDecimal[] objects depending on getCardinality().

Deprecated As of 1.1.

105.13.2.2 public static final int BIGINTEGER = 9

The BIGINTEGER (9) type. Attributes of this type should be stored as BigInteger, Vector with BigInteger or BigInteger[] objects, depending on the getCardinality() value.

Deprecated As of 1.1.

105.13.2.3 public static final int BOOLEAN = 11

The BOOLEAN (11) type. Attributes of this type should be stored as Boolean, Vector with Boolean or boolean[] objects depending on getCardinality().

105.13.2.4 public static final int BYTE = 6

The BYTE (6) type. Attributes of this type should be stored as Byte, Vector with Byte or byte[] objects, depending on the getCardinality() value.

105.13.2.5 public static final int CHARACTER = 5

The CHARACTER (5) type. Attributes of this type should be stored as Character, Vector with Character or char[] objects, depending on the getCardinality() value.

105.13.2.6 public static final int DOUBLE = 7

The DOUBLE (7) type. Attributes of this type should be stored as Double, Vector with Double or double[] objects, depending on the getCardinality() value.

105.13.2.7 public static final int FLOAT = 8

The FLOAT (8) type. Attributes of this type should be stored as Float, Vector with Float or float[] objects, depending on the getCardinality() value.

105.13.2.8 public static final int INTEGER = 3

The INTEGER (3) type. Attributes of this type should be stored as Integer, Vector with Integer or int[] objects, depending on the getCardinality() value.

105.13.2.9 public static final int LONG = 2

The LONG (2) type. Attributes of this type should be stored as Long, Vector with Long or long[] objects, depending on the getCardinality() value.
105.13.2.10  public static final int PASSWORD = 12

The PASSWORD (12) type. Attributes of this type must be stored as String, Vector with String or String[] objects depending on (link getCardinality()). A PASSWORD must be treated as a string but the type can be used to disguise the information when displayed to a user to prevent others from seeing it.

Since 1.2

105.13.2.11  public static final int SHORT = 4

The SHORT (4) type. Attributes of this type should be stored as Short, Vector with Short or short[] objects, depending on the getCardinality() value.

105.13.2.12  public static final int STRING = 1

The STRING (1) type.

Attributes of this type should be stored as String, Vector with String or String[] objects, depending on the getCardinality() value.

105.13.2.13  public int getCardinality()

□ Return the cardinality of this attribute. The OSGi environment handles multi valued attributes in arrays ([]) or in Vector objects. The return value is defined as follows:

\[
\begin{align*}
  x &= \text{Integer.MIN_VALUE} & \text{no limit, but use Vector} \\
  x &< 0 & -x = \text{max occurrences, store in Vector} \\
  x &> 0 & x = \text{max occurrences, store in array []} \\
  x &= \text{Integer.MAX_VALUE} & \text{no limit, but use array []} \\
  x &= 0 & 1 \text{ occurrence required}
\end{align*}
\]

Returns The cardinality of this attribute.

105.13.2.14  public String[] getDefaultValue()

□ Return a default for this attribute. The object must be of the appropriate type as defined by the cardinality and getType(). The return type is a list of String objects that can be converted to the appropriate type. The cardinality of the return array must follow the absolute cardinality of this type. E.g. if the cardinality = 0, the array must contain 1 element. If the cardinality is 1, it must contain 0 or 1 elements. If it is -5, it must contain from 0 to max 5 elements. Note that the special case of a 0 cardinality, meaning a single value, does not allow arrays or vectors of 0 elements.

Returns Return a default value or null if no default exists.

105.13.2.15  public String getDescription()

□ Return a description of this attribute. The description may be localized and must describe the semantics of this type and any constraints.

Returns The localized description of the definition.

105.13.2.16  public String getID()

□ Unique identity for this attribute. Attributes share a global namespace in the registry. E.g. an attribute cn or commonName must always be a String and the semantics are always a name of some object. They share this aspect with LDAP/X.500 attributes. In these standards the OSI Object Identifier (OID) is used to uniquely identify an attribute. If such an OID exists, (which can be requested at several standard organisations and many companies already have a node in the tree) it can be returned here. Otherwise, a unique id should be returned which can be a Java class name (reverse domain name) or generated with a GUID algorithm. Note that all LDAP defined attributes already have
an OID. It is strongly advised to define the attributes from existing LDAP schemes which will give
the OID. Many such schemes exist ranging from postal addresses to DHCP parameters.

Returns The id or oid

105.13.2.17 public String getName()
□ Get the name of the attribute. This name may be localized.

Returns The localized name of the definition.

105.13.2.18 public String[] getOptionLabels()
□ Return a list of labels of option values.

The purpose of this method is to allow menus with localized labels. It is associated with getOption-
Values. The labels returned here are ordered in the same way as the values in that method.

If the function returns null, there are no option labels available.

This list must be in the same sequence as the getOptionValues() method. I.e. for each index i in
getOptionLabels, i in getOptionValues() should be the associated value.

For example, if an attribute can have the value male, female, unknown, this list can return (for
dutch) new String[] { "Man", "Vrouw", "Onbekend" }.

Returns A list values

105.13.2.19 public String[] getOptionValues()
□ Return a list of option values that this attribute can take.

If the function returns null, there are no option values available.

Each value must be acceptable to validate() (return "") and must be a String object that can be con-
verted to the data type defined by getType() for this attribute.

This list must be in the same sequence as getOptionLabels(). I.e. for each index i in getOptionVal-
ues, i in getOptionLabels() should be the label.

For example, if an attribute can have the value male, female, unknown, this list can return new
String[] { "male", "female", "unknown" }.

Returns A list values

105.13.2.20 public int getType()
□ Return the type for this attribute.

Defined in the following constants which map to the appropriate Java type. STRING, LONG, INTEGER,
CHAR, BYTE, DOUBLE, FLOAT, BOOLEAN.

Returns The type for this attribute.

105.13.2.21 public String validate(String value)
value The value before turning it into the basic data type. If the cardinality indicates a multi-valued at-
ttribute then the given string must be escaped.

□ Validate an attribute in String form. An attribute might be further constrained in value. This
method will attempt to validate the attribute according to these constraints. It can return three dif-
ferent values:

null No validation present
"" No problems detected
"..." A localized description of why the value is wrong
If the cardinality of this attribute is multi-valued then this string must be interpreted as a comma delimited string. The complete value must be trimmed from white space as well as spaces around commas. Commas (',\u002C) and spaces ('\u0020') and backslashes ('\u005C') can be escaped with another backslash. Escaped spaces must not be trimmed. For example:

```plaintext
value=" a\,b,b|,c\, c\,d  " => [ "a,b", "b,c", " c", "d" ]
```

Returns null, "", or another string.

**105.13.3**  
public interface MetaTypeInformation extends MetaTypeProvider

A MetaType Information object is created by the MetaTypeService to return meta type information for a specific bundle.

Since 1.1

Concurrency Thread-safe

No Implement Consumers of this API must not implement this interface

**105.13.3.1**  
public Bundle getBundle()

Return the bundle for which this object provides meta type information.

Returns Bundle for which this object provides meta type information.

**105.13.3.2**  
public String[] getFactoryPids()

Return the Factory PIDs (for ManagedServiceFactories) for which ObjectClassDefinition information is available.

Returns Array of Factory PIDs.

**105.13.3.3**  
public String[] getPids()

Return the PIDs (for ManagedServices) for which ObjectClassDefinition information is available.

Returns Array of PIDs.

**105.13.4**  
public interface MetaTypeProvider

Provides access to metatypes. This interface can be implemented on a Managed Service or Managed Service Factory as well as registered as a service. When registered as a service, it must be registered with a METATYPE_FACTORY_PID or METATYPE_PID service property (or both). Any PID mentioned in either of these factories must be a valid argument to the getObjectClassDefinition(String, String) method.

Concurrency Thread-safe

**105.13.4.1**  
public static final String METATYPE_FACTORY_PID = "metatype.factory.pid"

Service property to signal that this service has ObjectClassDefinition objects for the given factory PIDs. The type of this service property is String+.

Since 1.2

**105.13.4.2**  
public static final String METATYPE_PID = "metatype.pid"

Service property to signal that this service has ObjectClassDefinition objects for the given PIDs. The type of this service property is String+.

Since 1.2
**105.13.3**

`public String[] getLocales()`

- Return a list of available locales. The results must be names that consist of language `_[_country_[_variation]]` as is customary in the Locale class.

*Returns* An array of locale strings or null if there is no locale specific localization can be found.

**105.13.4.4**

`public ObjectClassDefinition get ObjectClassDefinition(String id, String locale)`

- `id` The ID of the requested object class. This can be a pid or factory pid returned by getPids or getFactoryPids.
- `locale` The locale of the definition or null for default locale.

- Returns an object class definition for the specified id localized to the specified locale.

The locale parameter must be a name that consists of language `_[_country_[_variation]]` as is customary in the Locale class. This Locale class is not used because certain profiles do not contain it.

*Returns* A `ObjectClassDefinition` object.

*Throws* `IllegalArgumentException` – If the id or locale arguments are not valid

**105.13.5**

`public interface MetaTypeService`

The MetaType Service can be used to obtain meta type information for a bundle. The MetaType Service will examine the specified bundle for meta type documents to create the returned `MetaTypeInformation` object.

If the specified bundle does not contain any meta type documents, then a `MetaTypeInformation` object will be returned that wraps any `ManagedService` or `ManagedServiceFactory` services registered by the specified bundle that implement `MetaTypeProvider`. Thus the MetaType Service can be used to retrieve meta type information for bundles which contain a meta type documents or which provide their own `MetaTypeProvider` objects.

*Since* 1.1

*Concurrency* Thread safe

*No Implement* Consumers of this API must not implement this interface

**105.13.5.1**

`public static final String METATYPE_DOCUMENTS_LOCATION = "OSGI-INF/metatype"`

Location of meta type documents. The MetaType Service will process each entry in the meta type documents directory.

**105.13.5.2**

`public MetaTypeInformation getMetaTypeInformation(Bundle bundle)`

- `bundle` The bundle for which meta type information is requested.

- Returns A `MetaTypeInformation` object for the specified bundle.

**105.13.6**

`public interface ObjectClassDefinition`

Description for the data type information of an objectclass.

*Concurrency* Thread safe

**105.13.6.1**

`public static final int ALL = -1`

Argument for `getAttributeDefinitions(int)`. ALL indicates that all the definitions are returned. The value is -1.

**105.13.6.2**

`public static final int OPTIONAL = 2`

Argument for `getAttributeDefinitions(int)`. 
105.13.6.3  public static final int REQUIRED = 1
Argument for getAttributeDefinitions(int).
REQUIRED indicates that only the required definitions are returned. The value is 1.

105.13.6.4  public AttributeDefinition[] getAttributeDefinitions(int filter)

filter  ALL,REQUIRED,OPTIONAL

□  Return the attribute definitions for this object class.
Return a set of attributes. The filter parameter can distinguish between ALL,REQUIRED or the OPTIONAL attributes.

Returns  An array of attribute definitions or null if no attributes are selected

105.13.6.5  public String getDescription()

□  Return a description of this object class. The description may be localized.

Returns  The description of this object class.

105.13.6.6  public InputStream getIcon(int size) throws IOException

size  Requested size of an icon, e.g. a 16x16 pixels icon then size = 16

□  Return an InputStream object that can be used to create an icon from.
Indicate the size and return an InputStream object containing an icon. The returned icon maybe larger or smaller than the indicated size.
The icon may depend on the localization.

Returns  An InputStream representing an icon or null
Throws  IOException – If the InputStream cannot be returned.

105.13.6.7  public String getID()

□  Return the id of this object class.
ObjectDefinition objects share a global namespace in the registry. They share this aspect with LDAP/ X.500 attributes. In these standards the OSI Object Identifier (OID) is used to uniquely identify object classes. If such an OID exists, (which can be requested at several standard organisations and many companies already have a node in the tree) it can be returned here. Otherwise, a unique id should be returned which can be a java class name (reverse domain name) or generated with a GUID algorithm. Note that all LDAP defined object classes already have an OID associated. It is strongly advised to define the object classes from existing LDAP schemes which will give the OID for free. Many such schemes exist ranging from postal addresses to DHCP parameters.

Returns  The id of this object class.

105.13.6.8  public String getName()

□  Return the name of this object class. The name may be localized.

Returns  The name of this object class.

105.14  References

[1]  LDAP.
[2] Understanding and Deploying LDAP Directory services
User Admin Service Specification

Version 1.1

107.1 Introduction

OSGi frameworks are often used in places where end users or devices initiate actions. These kinds of actions inevitably create a need for authenticating the initiator. Authenticating can be done in many different ways, including with passwords, one-time token cards, bio-metrics, and certificates.

Once the initiator is authenticated, it is necessary to verify that this principal is authorized to perform the requested action. This authorization can only be decided by the operator of the OSGi environment, and thus requires administration.

The User Admin service provides this type of functionality. Bundles can use the User Admin service to authenticate an initiator and represent this authentication as an Authorization object. Bundles that execute actions on behalf of this user can use the Authorization object to verify if that user is authorized.


107.1.1 Essentials

- **Authentication** - A large number of authentication schemes already exist, and more will be developed. The User Admin service must be flexible enough to adapt to the many different authentication schemes that can be run on a computer system.
- **Authorization** - All bundles should use the User Admin service to authenticate users and to find out if those users are authorized. It is therefore paramount that a bundle can find out authorization information with little effort.
- **Security** - Detailed security, based on the Framework security model, is needed to provide safe access to the User Admin service. It should allow limited access to the credentials and other properties.
- **Extensibility** - Other bundles should be able to build on the User Admin service. It should be possible to examine the information from this service and get real-time notifications of changes.
- **Properties** - The User Admin service must maintain a persistent database of users. It must be possible to use this database to hold more information about this user.
- **Administration** - Administering authorizations for each possible action and initiator is time-consuming and error-prone. It is therefore necessary to have mechanisms to group end users and make it simple to assign authorizations to all members of a group at one time.

107.1.2 Entities

This Specification defines the following User Admin service entities:

- **User Admin** - This interface manages a database of named roles which can be used for authorization and authentication purposes.
- **Role** - This interface exposes the characteristics shared by all roles: a name, a type, and a set of properties.
- **User** - This interface (which extends Role) is used to represent any entity which may have credentials associated with it. These credentials can be used to authenticate an initiator.
- **Group** - This interface (which extends User) is used to contain an aggregation of named Role objects (Group or User objects).
- **Authorization** - This interface encapsulates an authorization context on which bundles can base authorization decisions.
- **User Admin Event** - This class is used to represent a role change event.
- **User Admin Listener** - This interface provides a listener for events of type UserAdminEvent that can be registered as a service.
- **User Admin Permission** - This permission is needed to configure and access the roles managed by a User Admin service.
- **Role.USER_ANYONE** - This is a special User object that represents any user, it implies all other User objects. It is also used when a Group is used with only basic members. The Role.USER_ANYONE is then the only required member.

### Figure 107.1
User Admin Service, org.osgi.service.useradmin

#### 107.1.3 Operation

An Operator uses the User Admin service to define OSGi framework users and configure them with properties, credentials, and roles.

A Role object represents the initiator of a request (human or otherwise). This specification defines two types of roles:
User Admin Service Specification Version 1.1

Authentication

- **User** - A User object can be configured with credentials, such as a password, and properties, such as address, telephone number, and so on.
- **Group** - A Group object is an aggregation of basic and required roles. Basic and required roles are used in the authorization phase.

An OSGi framework can have several entry points, each of which will be responsible for authenticating incoming requests. An example of an entry point is the Http Service, which delegates authentication of incoming requests to the handleSecurity method of the HttpContext object that was specified when the target servlet or resource of the request was registered.

The OSGi framework entry points should use the information in the User Admin service to authenticate incoming requests, such as a password stored in the private credentials or the use of a certificate.

A bundle can determine if a request for an action is authorized by looking for a Role object that has the name of the requested action.

The bundle may execute the action if the Role object representing the initiator implies the Role object representing the requested action.

For example, an initiator Role object X implies an action Group object A if:

- X implies at least one of A's basic members, and
- X implies all of A's required members.

An initiator Role object X implies an action User object A if:

- A and X are equal.

The Authorization class handles this non-trivial logic. The User Admin service can capture the privileges of an authenticated User object into an Authorization object. The Authorization.hasRole method checks if the authenticate User object has (or implies) a specified action Role object.

For example, in the case of the Http Service, the HttpContext object can authenticate the initiator and place an Authorization object in the request header. The servlet calls the hasRole method on this Authorization object to verify that the initiator has the authority to perform a certain action. See **Authentication** on page 54.

### 107.2 Authentication

The authentication phase determines if the initiator is actually the one it says it is. Mechanisms to authenticate always need some information related to the user or the OSGi framework to authenticate an external user. This information can consist of the following:

- A secret known only to the initiator.
- Knowledge about cards that can generate a unique token.
- Public information like certificates of trusted signers.
- Information about the user that can be measured in a trusted way.
- Other specific information.

### 107.2.1 Repository

The User Admin service offers a repository of Role objects. Each Role object has a unique name and a set of properties that are readable by anyone, and are changeable when the changer has the UserAdminPermission. Additionally, User objects, a sub-interface of Role, also have a set of private protected properties called credentials. Credentials are an extra set of properties that are used to authenticate users and that are protected by UserAdminPermission.
Properties are accessed with the `Role.getProperties()` method and credentials with the `User.getCredentials()` method. Both methods return a `Dictionary` object containing key/value pairs. The keys are `String` objects and the values of the `Dictionary` object are limited to `String` or `byte[]` objects.

This specification does not define any standard keys for the properties or credentials. The keys depend on the implementation of the authentication mechanism and are not formally defined by OSGi specifications.

The repository can be searched for objects that have a unique property (key/value pair) with the method `UserAdmin.getUser(String, String)`. This makes it easy to find a specific user related to a specific authentication mechanism. For example, a secure card mechanism that generates unique tokens could have a serial number identifying the user. The owner of the card could be found with the method

```java
User owner = useradmin.getUser(
    "secure-card-serial", "132456712-1212" );
```

If multiple `User` objects have the same property (key and value), a null is returned.

There is a convenience method to verify that a user has a credential without actually getting the credential. This is the `User.hasCredential(String, Object)` method.

Access to credentials is protected on a name basis by `UserAdminPermission`. Because properties can be read by anyone with access to a `User` object, `UserAdminPermission` only protects change access to properties.

### 107.2.2 Basic Authentication

The following example shows a very simple authentication algorithm based on passwords.

The vendor of the authentication bundle uses the property "com.acme.basic-id" to contain the name of a user as it logs in. This property is used to locate the `User` object in the repository. Next, the credential "com.acme.password" contains the password and is compared to the entered password. If the password is correct, the `User` object is returned. In all other cases a `SecurityException` is thrown.

```java
public User authenticate(
    UserAdmin ua, String name, String pwd )
    throws SecurityException {  
    User user = ua.getUser("com.acme.basicid", username);
    if (user == null)
        throw new SecurityException( "No such user" );
    if (!user.hasCredential("com.acme.password", pwd))
        throw new SecurityException( 
            "Invalid password" );
    return user;
}
```

### 107.2.3 Certificates

Authentication based on certificates does not require a shared secret. Instead, a certificate contains a name, a public key, and the signature of one or more signers.

The name in the certificate can be used to locate a `User` object in the repository. Locating a `User` object, however, only identifies the initiator and does not authenticate it.

1. The first step to authenticate the initiator is to verify that it has the private key of the certificate.
2. Next, the User Admin service must verify that it has a User object with the right property, for example "com.acme.certificate"="Fudd".

3. The next step is to see if the certificate is signed by a trusted source. The bundle could use a central list of trusted signers and only accept certificates signed by those sources. Alternatively, it could require that the certificate itself is already stored in the repository under a unique key as a byte[] in the credentials.

4. In any case, once the certificate is verified, the associated User object is authenticated.

107.3 Authorization

The User Admin service authorization architecture is a role-based model. In this model, every action that can be performed by a bundle is associated with a role. Such a role is a Group object (called group from now on) from the User Admin service repository. For example, if a servlet could be used to activate the alarm system, there should be a group named AlarmSystemActivation.

The operator can administrate authorizations by populating the group with User objects (users) and other groups. Groups are used to minimize the amount of administration required. For example, it is easier to create one Administrators group and add administrative roles to it rather than individually administer all users for each role. Such a group requires only one action to remove or add a user as an administrator.

The authorization decision can now be made in two fundamentally different ways:

An initiator could be allowed to carry out an action (represented by a Group object) if it implied any of the Group object's members. For example, the AlarmSystemActivation Group object contains an Administrators group and a Family Group object:

- Administrators = { Elmer, Pepe, Bugs }
- Family = { Elmer, Pepe, Daffy }
- AlarmSystemActivation = { Administrators, Family }

Any of the four members Elmer, Pepe, Daffy, or Bugs can activate the alarm system.

Alternatively, an initiator could be allowed to perform an action (represented by a Group object) if it implied all the Group object's members. In this case, using the same AlarmSystemActivation group, only Elmer and Pepe would be authorized to activate the alarm system, since Daffy and Bugs are not members of both the Administrators and Family Group objects.

The User Admin service supports a combination of both strategies by defining both a set of basic members (any) and a set of required members (all).

- Administrators = { Elmer, Pepe, Bugs }
- Family = { Elmer, Pepe, Daffy }
- AlarmSystemActivation
  - required = { Administrators }
  - basic = { Family }

The difference is made when Role objects are added to the Group object. To add a basic member, use the Group.addMember(Role) method. To add a required member, use the Group.addRequiredMember(Role) method.

Basic members define the set of members that can get access and required members reduce this set by requiring the initiator to imply each required member.

A User object implies a Group object if it implies the following:
Authorization

- All of the Group's required members, and
- At least one of the Group's basic members

A User object always implies itself.

If only required members are used to qualify the implication, then the standard user Role.USER_ANYONE can be obtained from the User Admin service and added to the Group object. This Role object is implied by anybody and therefore does not affect the required members.

107.3.1 The Authorization Object

The complexity of authorization is hidden in an Authorization class. Normally, the authenticator should retrieve an Authorization object from the User Admin service by passing the authenticated User object as an argument. This Authorization object is then passed to the bundle that performs the action. This bundle checks the authorization with the Authorization.hasRole(String) method. The performing bundle must pass the name of the action as an argument. The Authorization object checks whether the authenticated user implies the Role object, specifically a Group object, with the given name. This is shown in the following example.

```java
public void activateAlarm(Authorization auth) {
    if ( auth.hasRole( "AlarmSystemActivation" ) ) {
        // activate the alarm
        ...
    } else throw new SecurityException(
        "Not authorized to activate alarm" );
}
```

107.3.2 Authorization Example

This section demonstrates a possible use of the User Admin service. The service has a flexible model and many other schemes are possible.

Assume an Operator installs an OSGi framework. Bundles in this environment have defined the following action groups:

- AlarmSystemControl
- InternetAccess
- TemperatureControl
- PhotoAlbumEdit
- PhotoAlbumView
- PortForwarding

Installing and uninstalling bundles could potentially extend this set. Therefore, the Operator also defines a number of groups that can be used to contain the different types of system users.

- Administrators
- Buddies
- Children
- Adults
- Residents

In a particular instance, the Operator installs it in a household with the following residents and buddies:

- Residents: Elmer, Fudd, Marvin, Pepe
- Buddies: Daffy, Foghorn

First, the residents and buddies are assigned to the system user groups. Second, the user groups need to be assigned to the action groups.
The following tables show how the groups could be assigned.

**Table 107.1**  
*Example Groups with Basic and Required Members*

<table>
<thead>
<tr>
<th>Groups</th>
<th>Elmer</th>
<th>Fudd</th>
<th>Marvin</th>
<th>Pepe</th>
<th>Daffy</th>
<th>Foghorn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Residents</td>
<td>Basic</td>
<td>Basic</td>
<td>Basic</td>
<td>Basic</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Buddies</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Basic</td>
<td>Basic</td>
<td></td>
</tr>
<tr>
<td>Children</td>
<td>-</td>
<td>-</td>
<td>Basic</td>
<td>Basic</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Adults</td>
<td>Basic</td>
<td>Basic</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Administrators</td>
<td>Basic</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 107.2**  
*Example Action Groups with their Basic and Required Members*

<table>
<thead>
<tr>
<th>Groups</th>
<th>Residents</th>
<th>Buddies</th>
<th>Children</th>
<th>Adults</th>
<th>Admin</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlarmSystemControl</td>
<td>Basic</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Required</td>
</tr>
<tr>
<td>InternetAccess</td>
<td>Basic</td>
<td>-</td>
<td>-</td>
<td>Required</td>
<td>-</td>
</tr>
<tr>
<td>TemperatureControl</td>
<td>Basic</td>
<td>-</td>
<td>Required</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PhotoAlbumEdit</td>
<td>Basic</td>
<td>-</td>
<td>Basic</td>
<td>Basic</td>
<td>-</td>
</tr>
<tr>
<td>PhotoAlbumView</td>
<td>Basic</td>
<td>Basic</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PortForwarding</td>
<td>Basic</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Required</td>
</tr>
</tbody>
</table>

## 107.4 Repository Maintenance

The UserAdmin interface is a straightforward API to maintain a repository of User and Group objects. It contains methods to create new Group and User objects with the `createRole(String,int)` method. The method is prepared so that the same signature can be used to create new types of roles in the future. The interface also contains a method to remove a Role object.

The existing configuration can be obtained with methods that list all Role objects using a filter argument. This filter, which has the same syntax as the Framework filter, must only return the Role objects for which the filter matches the properties.

Several utility methods simplify getting User objects depending on their properties.

## 107.5 User Admin Events

Changes in the User Admin service can be determined in real time. Each User Admin service implementation must send a `UserAdminEvent` object to any service in the Framework service registry that is registered under the `UserAdminListener` interface. This event must be send asynchronously from the cause of the event. The way events must be delivered is the same as described in *Delivering Events of OSGi Core Release 6*.

This procedure is demonstrated in the following code sample.

```java
class Listener implements UserAdminListener{
    public void roleChanged( UserAdminEvent event ) {
        ...
    }
}

class MyActivator implements BundleActivator {
    public void start( BundleContext context ) {
        context.registerService(
            UserAdminListener.class.getName(),
            ...)
    }
...
new Listener(), null );
}
public void stop( BundleContext context ) {}}
}

It is not necessary to unregister the listener object when the bundle is stopped because the Framework automatically unregisters it. Once registered, the UserAdminListener object must be notified of all changes to the role repository.

### 107.5.1 Event Admin and User Admin Change Events

User admin events must be delivered asynchronously to the Event Admin service by the implementation, if present. The topic of a User Admin Event is:

```
org/osgi/service/useradmin/UserAdmin/<eventtype>
```

The following event types are supported:

- ROLE_CREATED
- ROLE_CHANGED
- ROLE_REMOVED

All User Admin Events must have the following properties:

- **event** - (UserAdminEvent) The event that was broadcast by the User Admin service.
- **role** - (Role) The role object that was created, modified or removed.
- **role.name** - (String) The name of the role.
- **role.type** - (Integer) One of ROLE, USER or GROUP.
- **service** - (ServiceReference) The Service Reference of the User Admin service.
- **service.id** - (Long) The User Admin service's ID.
- **service.objectClass** - (String[]) The User Admin service's object class (which must include org.osgi.service.useradmin.UserAdmin)
- **service.pid** - (String) The User Admin service's persistent identity

### 107.6 Security

The User Admin service is related to the security model of the OSGi framework, but is complementary to the [1] *The Java Security Architecture for JDK 1.2*. The final permission of most code should be the intersection of the Java 2 Permissions, which are based on the code that is executing, and the User Admin service authorization, which is based on the user for whom the code runs.

#### 107.6.1 User Admin Permission

The User Admin service defines the UserAdminPermission class that can be used to restrict bundles in accessing credentials. This permission class has the following actions:

- **changeProperty** - This permission is required to modify properties. The name of the permission is the prefix of the property name.
- **changeCredential** - This action permits changing credentials. The name of the permission is the prefix of the name of the credential.
- **getCredential** - This action permits getting credentials. The name of the permission is the prefix of the credential.

If the name of the permission is "admin", it allows the owner to administer the repository. No action is associated with the permission in that case.
Otherwise, the permission name is used to match the property name. This name may end with a ".*" string to indicate a wildcard. For example, com.acme.* matches com.acme.fudd.elmer and com.acme.bugs.

### 107.7 Relation to JAAS

At a glance, the Java Authorization and Authentication Service (JAAS) seems to be a very suitable model for user administration. The OSGi organization, however, decided to develop an independent User Admin service because JAAS was not deemed applicable. The reasons for this include dependency on Java SE version 1.3 ("JDK 1.3") and existing mechanisms in the previous OSGi Service Gateway 1.0 specification.

#### 107.7.1 JDK 1.3 Dependencies

The authorization component of JAAS relies on the `java.security.DomainCombiner` interface, which provides a means to dynamically update the `ProtectionDomain` objects affiliated with an `AccessControlContext` object.

This interface was added in JDK 1.3. In the context of JAAS, the `SubjectDomainCombiner` object, which implements the `DomainCombiner` interface, is used to update `ProtectionDomain` objects. The permissions of `ProtectionDomain` objects depend on where code came from and who signed it, with permissions based on who is running the code.

Leveraging JAAS would have resulted in user-based access control on the OSGi framework being available only with JDK 1.3, which was not deemed acceptable.

#### 107.7.2 Existing OSGi Mechanism

JAAS provides a pluggable authentication architecture, which enables applications and their underlying authentication services to remain independent from each other.

The Http Service already provides a similar feature by allowing servlet and resource registrations to be supported by an `HttpContext` object, which uses a callback mechanism to perform any required authentication checks before granting access to the servlet or resource. This way, the registering bundle has complete control on a per-servlet and per-resource basis over which authentication protocol to use, how the credentials presented by the remote requestor are to be validated, and who should be granted access to the servlet or resource.

#### 107.7.3 Future Road Map

In the future, the main barrier of 1.3 compatibility will be removed. JAAS could then be implemented in an OSGi environment. At that time, the User Admin service will still be needed and will provide complementary services in the following ways:

- The authorization component relies on group membership information to be stored and managed outside JAAS. JAAS does not manage persistent information, so the User Admin service can be a provider of group information when principals are assigned to a `Subject` object.
- The authorization component allows for credentials to be collected and verified, but a repository is needed to actually validate the credentials.

In the future, the User Admin service can act as the back-end database to JAAS. The only aspect JAAS will remove from the User Admin service is the need for the Authorization interface.

### 107.8 org.osgi.service.useradmin
User Admin Package Version 1.1.

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.useradmin; version="[1.1,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.useradmin; version="[1.1,1.2)"
```

### 107.8.1 Summary

- **Authorization** - The Authorization interface encapsulates an authorization context on which bundles can base authorization decisions, where appropriate.
- **Group** - A named grouping of roles (Role objects).
- **Role** - The base interface for Role objects managed by the User Admin service.
- **User** - A User role managed by a User Admin service.
- **UserAdmin** - This interface is used to manage a database of named Role objects, which can be used for authentication and authorization purposes.
- **UserAdminEvent** - Role change event.
- **UserAdminListener** - Listener for UserAdminEvents.
- **UserAdminPermission** - Permission to configure and access the Role objects managed by a User Admin service.

### 107.8.2 public interface Authorization

The Authorization interface encapsulates an authorization context on which bundles can base authorization decisions, where appropriate.

Bundles associate the privilege to access restricted resources or operations with roles. Before granting access to a restricted resource or operation, a bundle will check if the Authorization object passed to it possess the required role, by calling its `hasRole` method.

Authorization contexts are instantiated by calling the `UserAdmin.getAuthorization(User)` method.

**Trusting Authorization objects**

There are no restrictions regarding the creation of Authorization objects. Hence, a service must only accept Authorization objects from bundles that has been authorized to use the service using code based (or Java 2) permissions.

In some cases it is useful to use `ServicePermission` to do the code based access control. A service basing user access control on Authorization objects passed to it, will then require that a calling bundle has the `ServicePermission` to get the service in question. This is the most convenient way. The OSGi environment will do the code based permission check when the calling bundle attempts to get the service from the service registry.

Example: A servlet using a service on a user's behalf. The bundle with the servlet must be given the `ServicePermission` to get the Http Service.

However, in some cases the code based permission checks need to be more fine-grained. A service might allow all bundles to get it, but require certain code based permissions for some of its methods.

Example: A servlet using a service on a user’s behalf, where some service functionality is open to anyone, and some is restricted by code based permissions. When a restricted method is called (e.g., one handing over an Authorization object), the service explicitly checks that the calling bundle has permission to make the call.
No Implement Consumers of this API must not implement this interface

107.8.2.1 public String getName()

- Gets the name of the User that this Authorization context was created for.

Returns The name of the User object that this Authorization context was created for, or null if no user was specified when this Authorization context was created.

107.8.2.2 public String[] getRoles()

- Gets the names of all roles implied by this Authorization context.

Returns The names of all roles implied by this Authorization context, or null if no roles are in the context. The predefined role user.anyone will not be included in this list.

107.8.2.3 public boolean hasRole(String name)

name The name of the role to check for.

- Checks if the role with the specified name is implied by this Authorization context.

Bundles must define globally unique role names that are associated with the privilege of accessing restricted resources or operations. Operators will grant users access to these resources, by creating a Group object for each role and adding User objects to it.

Returns true if this Authorization context implies the specified role, otherwise false.

107.8.3 public interface Group

extends User

A named grouping of roles (Role objects).

Whether or not a given Authorization context implies a Group object depends on the members of that Group object.

A Group object can have two kinds of members: basic and required. A Group object is implied by an Authorization context if all of its required members are implied and at least one of its basic members is implied.

A Group object must contain at least one basic member in order to be implied. In other words, a Group object without any basic member roles is never implied by any Authorization context.

A User object always implies itself.

No loop detection is performed when adding members to Group objects, which means that it is possible to create circular implications. Loop detection is instead done when roles are checked. The semantics is that if a role depends on itself (i.e., there is an implication loop), the role is not implied.

The rule that a Group object must have at least one basic member to be implied is motivated by the following example:

```
group foo
   required members: marketing
   basic members: alice, bob
```

Privileged operations that require membership in “foo” can be performed only by “alice” and “bob”, who are in marketing.

If “alice” and “bob” ever transfer to a different department, anybody in marketing will be able to assume the “foo” role, which certainly must be prevented. Requiring that “foo” (or any Group object for that matter) must have at least one basic member accomplishes that.
However, this would make it impossible for a Group object to be implied by just its required members. An example where this implication might be useful is the following declaration: "Any citizen who is an adult is allowed to vote." An intuitive configuration of "voter" would be:

```
group voter
  required members: citizen, adult
  basic members:
```

However, according to the above rule, the "voter" role could never be assumed by anybody, since it lacks any basic members. In order to address this issue a predefined role named "user.anyone" can be specified, which is always implied. The desired implication of the "voter" group can then be achieved by specifying "user.anyone" as its basic member, as follows:

```
group voter
  required members: citizen, adult
  basic members: user.anyone
```

No Implement Consumers of this API must not implement this interface

107.8.3.1 public boolean addMember(Role role)

```
role The role to add as a basic member.

□ Adds the specified Role object as a basic member to this Group object.

Returns true if the given role could be added as a basic member, and false if this Group object already contains a Role object whose name matches that of the specified role.

Throws SecurityException – If a security manager exists and the caller does not have the UserAdminPermission with name admin.
```

107.8.3.2 public boolean addRequiredMember(Role role)

```
role The Role object to add as a required member.

□ Adds the specified Role object as a required member to this Group object.

Returns true if the given Role object could be added as a required member, and false if this Group object already contains a Role object whose name matches that of the specified role.

Throws SecurityException – If a security manager exists and the caller does not have the UserAdminPermission with name admin.
```

107.8.3.3 public Role[] getMembers()

```
□ Gets the basic members of this Group object.

Returns The basic members of this Group object, or null if this Group object does not contain any basic members.
```

107.8.3.4 public Role[] getRequiredMembers()

```
□ Gets the required members of this Group object.

Returns The required members of this Group object, or null if this Group object does not contain any required members.
```

107.8.3.5 public boolean removeMember(Role role)

```
role The Role object to remove from this Group object.

□ Removes the specified Role object from this Group object.
```
Returns true if the Role object could be removed, otherwise false.

Throws SecurityException – If a security manager exists and the caller does not have the UserAdminPermission with name admin.

**107.8.4**

**public interface Role**

The base interface for Role objects managed by the User Admin service.

This interface exposes the characteristics shared by all Role classes: a name, a type, and a set of properties.

Properties represent public information about the Role object that can be read by anyone. Specific UserAdminPermission objects are required to change a Role object’s properties.

Role object properties are Dictionary objects. Changes to these objects are propagated to the User Admin service and made persistent.

Every User Admin service contains a set of predefined Role objects that are always present and cannot be removed. All predefined Role objects are of type ROLE. This version of the org.osgi.service.useradmin package defines a single predefined role named “user.anyone”, which is inherited by any other role. Other predefined roles may be added in the future. Since “user.anyone” is a Role object that has properties associated with it that can be read and modified. Access to these properties and their use is application specific and is controlled using UserAdminPermission in the same way that properties for other Role objects are.

**No Implement** Consumers of this API must not implement this interface.

**107.8.4.1**

**public static final int GROUP = 2**

The type of a Group role.

The value of GROUP is 2.

**107.8.4.2**

**public static final int ROLE = 0**

The type of a predefined role.

The value of ROLE is 0.

**107.8.4.3**

**public static final int USER = 1**

The type of a User role.

The value of USER is 1.

**107.8.4.4**

**public static final String USER_ANYONE = "user.anyone"**

The name of the predefined role, user.anyone, that all users and groups belong to.

Since 1.1

**107.8.4.5**

**public String getName()**

Returns the name of this role.

**Returns** The role’s name.

**107.8.4.6**

**public Dictionary getProperties()**

Returns a Dictionary of the (public) properties of this Role object. Any changes to the returned Dictionary will change the properties of this Role object. This will cause a UserAdminEvent object of type UserAdminEvent.ROLE_CHANGED to be broadcast to any UserAdminListener objects.

Only objects of type String may be used as property keys, and only objects of type String or byte[] may be used as property values. Any other types will cause an exception of type IllegalArgumentException to be raised.
In order to add, change, or remove a property in the returned Dictionary, a UserAdminPermission named after the property name (or a prefix of it) with action changeProperty is required.

Returns Dictionary containing the properties of this Role object.

107.8.4.7 public int getType()

- Returns the type of this role.

Returns The role's type.

107.8.5 public interface User
extends Role

A User role managed by a User Admin service.

In this context, the term "user" is not limited to just human beings. Instead, it refers to any entity that may have any number of credentials associated with it that it may use to authenticate itself.

In general, User objects are associated with a specific User Admin service (namely the one that created them), and cannot be used with other User Admin services.

A User object may have credentials (and properties, inherited from the Role class) associated with it. Specific UserAdminPermission objects are required to read or change a User object's credentials.

Credentials are Dictionary objects and have semantics that are similar to the properties in the Role class.

No Implement Consumers of this API must not implement this interface

107.8.5.1 public Dictionary getCredentials()

- Returns a Dictionary of the credentials of this User object. Any changes to the returned Dictionary object will change the credentials of this User object. This will cause a UserAdminEvent object of type UserAdminEvent.ROLE_CHANGED to be broadcast to any UserAdminListeners objects.

Only objects of type String may be used as credential keys, and only objects of type String or of type byte[] may be used as credential values. Any other types will cause an exception of type IllegalArgumentException to be raised.

In order to retrieve a credential from the returned Dictionary object, a UserAdminPermission named after the credential name (or a prefix of it) with action getCredential is required.

In order to add or remove a credential from the returned Dictionary object, a UserAdminPermission named after the credential name (or a prefix of it) with action changeCredential is required.

Returns Dictionary object containing the credentials of this User object.

107.8.5.2 public boolean hasCredential(String key, Object value)

key The credential key.
value The credential value.

- Checks to see if this User object has a credential with the specified key set to the specified value.

If the specified credential value is not of type String or byte[], it is ignored, that is, false is returned (as opposed to an IllegalArgumentException being raised).

Returns true if this user has the specified credential; false otherwise.

Throws SecurityException – If a security manager exists and the caller does not have the UserAdminPermission named after the credential key (or a prefix of it) with action getCredential.
107.8.6 **public interface UserAdmin**

This interface is used to manage a database of named `Role` objects, which can be used for authentication and authorization purposes.

This version of the User Admin service defines two types of `Role` objects: "User" and "Group". Each type of role is represented by an `int` constant and an interface. The range of positive integers is reserved for new types of roles that may be added in the future. When defining proprietary role types, negative constant values must be used.

Every role has a name and a type.

A `User` object can be configured with credentials (e.g., a password) and properties (e.g., a street address, phone number, etc.).

A `Group` object represents an aggregation of `User` and `Group` objects. In other words, the members of a `Group` object are roles themselves.

Every User Admin service manages and maintains its own namespace of `Role` objects, in which each `Role` object has a unique name.

No Implement Consumers of this API must not implement this interface

107.8.6.1 **public Role createRole(String name, int type)**

- `name`  The name of the `Role` object to create.
- `type`  The type of the `Role` object to create. Must be either a `Role.USER` type or `Role.GROUP` type.

□ Creates a `Role` object with the given name and of the given type.

If a `Role` object was created, a `UserAdminEvent` object of type `UserAdminEvent.ROLE_CREATED` is broadcast to any `UserAdminListener` object.

*Returns* The newly created `Role` object, or null if a role with the given name already exists.

*Throws* `IllegalArgumentException` – if `type` is invalid.

SecurityException – If a security manager exists and the caller does not have the `UserAdminPermission` with name `admin`.

107.8.6.2 **public Authorization getAuthorization(User user)**

- `user`  The `User` object to create an `Authorization` object for, or null for the anonymous user.

□ Creates an `Authorization` object that encapsulates the specified `User` object and the `Role` objects it possesses. The null user is interpreted as the anonymous user. The anonymous user represents a user that has not been authenticated. An `Authorization` object for an anonymous user will be unnamed, and will only imply groups that `user.anyone` implies.

*Returns* the `Authorization` object for the specified `User` object.

107.8.6.3 **public Role getRole(String name)**

- `name`  The name of the `Role` object to get.

□ Gets the `Role` object with the given name from this User Admin service.

*Returns* The requested `Role` object, or null if this User Admin service does not have a `Role` object with the given name.

107.8.6.4 **public Role[] getRoles(String filter) throws InvalidSyntaxException**

- `filter`  The filter criteria to match.

□ Gets the `Role` objects managed by this User Admin service that have properties matching the specified LDAP filter criteria. See `org.osgi.framework.Filter` for a description of the filter syntax. If a null filter is specified, all `Role` objects managed by this User Admin service are returned.
Returns The Role objects managed by this User Admin service whose properties match the specified filter criteria, or all Role objects if a null filter is specified. If no roles match the filter, null will be returned.

Throws InvalidSyntaxException – If the filter is not well formed.

107.8.6.5 

**public User getUser(String key, String value)**

- **key** The property key to look for.
- **value** The property value to compare with.

● Gets the user with the given property value pair from the User Admin service database. This is a convenience method for retrieving a User object based on a property for which every User object is supposed to have a unique value (within the scope of this User Admin service), such as for example a X.500 distinguished name.

Returns A matching user, if exactly one is found. If zero or more than one matching users are found, null is returned.

107.8.6.6 

**public boolean removeRole(String name)**

- **name** The name of the Role object to remove.

● Removes the Role object with the given name from this User Admin service and all groups it is a member of.

If the Role object was removed, a UserAdminEvent object of type UserAdminEvent.ROLE_REMOVED is broadcast to any UserAdminListener object.

Returns true If a Role object with the given name is present in this User Admin service and could be removed, otherwise false.

Throws SecurityException – If a security manager exists and the caller does not have the UserAdminPermission with name admin.

107.8.7 

**public class UserAdminEvent**

Role change event.

UserAdminEvent objects are delivered asynchronously to any UserAdminListener objects when a change occurs in any of the Role objects managed by a User Admin service.

A type code is used to identify the event. The following event types are defined: ROLE_CREATED type, ROLE_CHANGED type, and ROLE_REMOVED type. Additional event types may be defined in the future.

See Also UserAdmin, UserAdminListener

107.8.7.1 

**public static final int ROLE_CHANGED = 2**

A Role object has been modified.

The value of ROLE_CHANGED is 0x00000002.

107.8.7.2 

**public static final int ROLE_CREATED = 1**

A Role object has been created.

The value of ROLE_CREATED is 0x00000001.

107.8.7.3 

**public static final int ROLE_REMOVED = 4**

A Role object has been removed.

The value of ROLE_REMOVED is 0x00000004.
107.8.7.4  public UserAdminEvent(ServiceReference ref, int type, Role role)

ref  The ServiceReference object of the User Admin service that generated this event.

type  The event type.

role  The Role object on which this event occurred.

□ Constructs a UserAdminEvent object from the given ServiceReference object, event type, and Role object.

107.8.7.5  public Role getRole()

□ Gets the Role object this event was generated for.

Returns  The Role object this event was generated for.

107.8.7.6  public ServiceReference getServiceReference()

□ Gets the ServiceReference object of the User Admin service that generated this event.

Returns  The User Admin service's ServiceReference object.

107.8.7.7  public int getType()

□ Returns the type of this event.

The type values are ROLE_CREATED type, ROLE_CHANGED type, and ROLE_REMOVED type.

Returns  The event type.

107.8.8  public interface UserAdminListener

Listener for UserAdminEvents.

UserAdminListener objects are registered with the Framework service registry and notified with a UserAdminEvent object when a Role object has been created, removed, or modified.

UserAdminListener objects can further inspect the received UserAdminEvent object to determine its type, the Role object it occurred on, and the User Admin service that generated it.

See Also  UserAdmin, UserAdminEvent

107.8.8.1  public void roleChanged(UserAdminEvent event)

event  The UserAdminEvent object.

□ Receives notification that a Role object has been created, removed, or modified.

107.8.9  public final class UserAdminPermission

extends BasicPermission

Permission to configure and access the Role objects managed by a User Admin service.

This class represents access to the Role objects managed by a User Admin service and their properties and credentials (in the case of User objects).

The permission name is the name (or name prefix) of a property or credential. The naming convention follows the hierarchical property naming convention. Also, an asterisk may appear at the end of the name, following a ".", or by itself, to signify a wildcard match. For example: "org.osgi.security.protocol.*" or "*" is valid, but "*protocol" or "a*b" are not valid.

The UserAdminPermission with the reserved name "admin" represents the permission required for creating and removing Role objects in the User Admin service, as well as adding and removing members in a Group object. This UserAdminPermission does not have any actions associated with it.
The actions to be granted are passed to the constructor in a string containing a list of one or more comma-separated keywords. The possible keywords are: changeProperty, changeCredential, and getCredential. Their meaning is defined as follows:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>changeProperty</td>
<td>Permission to change (i.e., add and remove) Role object properties whose names start with the name argument specified in the constructor.</td>
</tr>
<tr>
<td>changeCredential</td>
<td>Permission to change (i.e., add and remove) User object credentials whose names start with the name argument specified in the constructor.</td>
</tr>
<tr>
<td>getCredential</td>
<td>Permission to retrieve and check for the existence of User object credentials whose names start with the name argument specified in the constructor.</td>
</tr>
</tbody>
</table>

The action string is converted to lowercase before processing.

Following is a PermissionInfo style policy entry which grants a user administration bundle a number of UserAdminPermission object:

```
(org.osgi.service.useradmin.UserAdminPermission "admin")
(org.osgi.service.useradmin.UserAdminPermission "com.foo.*" "changeProperty, getCredential, changeCredential")
(org.osgi.service.useradmin.UserAdminPermission "user.*" "changeProperty, changeCredential")
```

The first permission statement grants the bundle the permission to perform any User Admin service operations of type “admin”, that is, create and remove roles and configure Group objects.

The second permission statement grants the bundle the permission to change any properties as well as get and change any credentials whose names start with com.foo.

The third permission statement grants the bundle the permission to change any properties and credentials whose names start with user.. This means that the bundle is allowed to change, but not retrieve any credentials with the given prefix.

The following policy entry empowers the Http Service bundle to perform user authentication:

```
grant codeBase "${jars}http.jar" {
    permission org.osgi.service.useradmin.UserAdminPermission "user.password", "getCredential";
};
```

The permission statement grants the Http Service bundle the permission to validate any password credentials (for authentication purposes), but the bundle is not allowed to change any properties or credentials.

### Concurrency

Thread-safe

107.8.9.1  
**public static final String ADMIN = "admin"**

The permission name “admin”.

107.8.9.2  
**public static final String CHANGE_CREDENTIAL = "changeCredential"**

The action string “changeCredential”.

107.8.9.3  
**public static final String CHANGE_PROPERTY = "changeProperty"**

The action string “changeProperty”.
107.8.9.4  public static final String GET_CREDENTIAL = "getCredential"

The action string "getCredential".

107.8.9.5  public UserAdminPermission(String name,String actions)

  name  the name of this UserAdminPermission
  actions  the action string.

  □ Creates a new UserAdminPermission with the specified name and actions. name is either the reserved string "admin" or the name of a credential or property, and actions contains a comma-separated list of the actions granted on the specified name. Valid actions are changeProperty, changeCredential, and getCredential.

  Throws  IllegalArgumentException—If name equals "admin" and actions are specified.

107.8.9.6  public boolean equals(Object obj)

  obj  the object to be compared for equality with this object.

  □ Checks two UserAdminPermission objects for equality. Checks that obj is a UserAdminPermission, and has the same name and actions as this object.

  Returns  true if obj is a UserAdminPermission object, and has the same name and actions as this UserAdmin-Permission object.

107.8.9.7  public String getActions()

  □ Returns the canonical string representation of the actions, separated by comma.

  Returns  the canonical string representation of the actions.

107.8.9.8  public int hashCode()

  □ Returns the hash code value for this object.

  Returns  A hash code value for this object.

107.8.9.9  public boolean implies(Permission p)

  p  the permission to check against.

  □ Checks if this UserAdminPermission object "implies" the specified permission.

  More specifically, this method returns true if:

  • p is an instance of UserAdminPermission,
  • p's actions are a proper subset of this object's actions, and
  • p's name is implied by this object's name. For example, "java.*" implies "java.home".

  Returns  true if the specified permission is implied by this object; false otherwise.

107.8.10  public PermissionCollection newPermissionCollection()

  □ Returns a new PermissionCollection object for storing UserAdminPermission objects.

  Returns  a new PermissionCollection object suitable for storing UserAdminPermission objects.

107.8.11  public String toString()

  □ Returns a string describing this UserAdminPermission object. This string must be in PermissionInfo encoded format.

  Returns  The PermissionInfo encoded string for this UserAdminPermission object.

See Also  org.osgi.service.permissionadmin.PermissionInfo.getEncoded()
107.9 References

[1] *The Java Security Architecture for JDK 1.2*
   Version 1.0, Sun Microsystems, October 1998

[2] *Java Authentication and Authorization Service*
## 110 Initial Provisioning Specification

### Version 1.2

#### 110.1 Introduction

To allow freedom regarding the choice of management protocol, the OSGi Specifications assumes an architecture to remotely manage an OSGi framework with a Management Agent. The Management Agent is implemented with a Management Bundle that can communicate with an unspecified management protocol.

This specification defines how the Management Agent can make its way to the OSGi framework, and gives a structured view of the problems and their corresponding resolution methods.

The purpose of this specification is to enable the management of an OSGi framework by an Operator, and (optionally) to hand over the management of the OSGi framework later to another Operator. This approach is in accordance with the OSGi remote management reference architecture.

This bootstrapping process requires the installation of a Management Agent, with appropriate configuration data, in the OSGi framework.

This specification consists of a prologue, in which the principles of the Initial Provisioning are outlined, and a number of mappings to different mechanisms.

#### 110.1.1 Essentials

- **Policy Free** - The proposed solution must be business model agnostic; none of the affected parties (Operators, SPS Manufacturers, etc.) should be forced into any particular business model.
- **Interoperability** - The Initial Provisioning must permit arbitrary inter-operability between management systems and OSGi frameworks. Any compliant Remote Manager should be able to manage any compliant OSGi framework, even in the absence of a prior business relationship. Adhering to this requirement allows a particular Operator to manage a variety of makes and models of OSGi framework Servers using a single management system of the Operator’s choice. This rule also gives the consumer the greatest choice when selecting an Operator.
- **Flexible** - The management process should be as open as possible, to allow innovation and specialization while still achieving interoperability.

#### 110.1.2 Entities

- **Provisioning Service** - A service registered with the Framework that provides information about the initial provisioning to the Management Agent.
- **Provisioning Dictionary** - A Dictionary object that is filled with information from the ZIP files that are loaded during initial setup.
- **RSH Protocol** - An OSGi specific secure protocol based on HTTP.
- **Management Agent** - A bundle that is responsible for managing an OSGi framework under control of a Remote Manager.
110.2 Procedure

The following procedure should be executed by an OSGi Framework implementation that supports this Initial Provisioning specification.

When the OSGi framework is first brought under management control, it must be provided with an initial request URL in order to be provisioned. Either the end user or the manufacturer may provide the initial request URL. How the initial request URL is transferred to the Framework is not specified, but a mechanism might, for example, be a command line parameter when the framework is started.

When asked to start the Initial Provisioning, the OSGi framework will send a request to the management system. This request is encoded in a URL, for example:

```
http://osgi.acme.com/remote-manager
```

This URL may use any protocol that is available on the OSGi framework Server. Many standard protocols exist, but it is also possible to use a proprietary protocol. For example, software could be present which can communicate with a smart card and could handle, for example, this URL:

```
smart-card://com1:0/7F20/6F38
```

Before the request URL is executed, the OSGi framework information is appended to the URL. This information includes at least the OSGi framework Identifier, but may also contain proprietary information, as long as the keys for this information do not conflict. Different URL schemes may use different methods of appending parameters; these details are specified in the mappings of this specification to concrete protocols.

The result of the request must be a ZIP file. (The content type should be application/zip). It is the responsibility of the underlying protocol to guarantee the integrity and authenticity of this ZIP file.

This ZIP file is unpacked and its entries (except bundle and bundle-url entries, described in Table 110.2) are placed in a Dictionary object. This Dictionary object is called the Provisioning Dictionary. It must be made available from the Provisioning Service in the service registry. The names of the entries in the ZIP file must not start with a solidus (`/`)
The ZIP file may contain only four types of dictionary entries: text, binary, bundle, or bundle-url. The type of an entry can be specified in different ways. An Initial Provisioning service must look in the following places to find the information about an entry’s (MIME) type (in the given order):

1. The manifest header InitialProvisioning-Entries of the given ZIP file. This header is defined in InitialProvisioning-Entries Manifest Header on page 153. If this header is present, but a given entry’s path is not named then try the next step.
2. The extension of the entry path name if one of .txt, .jar, .url extensions. See Table 110.1 on page 151 for the mapping of types, MIME types, and extensions.
3. The entry is assumed to be a binary type

The types can optionally be specified as a MIME type as defined in [7] MIME Types. The text and bundle-url entries are translated into a String object from an UTF-8 encoded byte array. All other entries must be stored as a byte[].

Table 110.1 Content types of provisioning ZIP file

<table>
<thead>
<tr>
<th>Type</th>
<th>MIME Type</th>
<th>Ext</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>MIME_STRING</td>
<td>.txt</td>
<td>Must be represented as a String object</td>
</tr>
<tr>
<td></td>
<td>text/plain; charset=utf-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>binary</td>
<td>MIME_BYTE_ARRAY</td>
<td>not txt, .url, or .jar</td>
<td>Must be represented as a byte array (byte[]).</td>
</tr>
<tr>
<td></td>
<td>application/octet-stream</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bundle</td>
<td>MIME_BUNDLE</td>
<td>.jar</td>
<td>Entries must be installed using BundleContext.installBundle(String, InputStream), with the InputStream object constructed from the contents of the ZIP entry. The location must be the name of the ZIP entry without leading solidus ('/' \u002F). This entry must not be stored in the Provisioning Dictionary. If a bundle with this location name is already installed in this system, then this bundle must be updated instead of installed. The MIME_BUNDLE_ALT version is intended for backward compatibility, it specifies the original MIME type for bundles before there was an official IANA MIME type. The content of this entry is a string coded in utf-8. Entries must be installed using BundleContext.installBundle(String, InputStream), with the InputStream object created from the given URL. The location must be the name of the ZIP entry without leading solidus ('/' \u002F). This entry must not be stored in the Provisioning Dictionary. If a bundle with this location URL is already installed in this system, then this bundle must be updated instead of installed.</td>
</tr>
<tr>
<td></td>
<td>application/vnd.osgi.bundle</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MIME_BUNDLE_ALT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>application/x-osgi-bundle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bundle-url</td>
<td>MIME_BUNDLE_URL</td>
<td>.url</td>
<td></td>
</tr>
<tr>
<td></td>
<td>text/x-osgi-bundle-url; charset=utf-8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Provisioning Service must install (but not start) all entries in the ZIP file that are typed with bundle or bundle-url.
If an entry named `PROVISIONING_START_BUNDLE` is present in the Provisioning Dictionary, then its content type must be text as defined in Table 110.1. The content of this entry must match the bundle location of a previously loaded bundle. This designated bundle must be given `AllPermission` and started.

If no `PROVISIONING_START_BUNDLE` entry is present in the Provisioning Dictionary, the Provisioning Dictionary should contain a reference to another ZIP file under the `PROVISIONING_REFERENCE` key. If both keys are absent, no further action must take place.

If this `PROVISIONING_REFERENCE` key is present and holds a `String` object that can be mapped to a valid URL, then a new ZIP file must be retrieved from this URL. The `PROVISIONING_REFERENCE` link may be repeated multiple times in successively loaded ZIP files.

Referring to a new ZIP file with such a URL allows a manufacturer to place a fixed reference inside the OSGi framework Server (in a file or smart card) that will provide some platform identifying information and then also immediately load the information from the management system. The `PROVISIONING_REFERENCE` link may be repeated multiple times in successively loaded ZIP files. The entry `PROVISIONING_UPDATE_COUNT` must be an `Integer` object that must be incremented on every iteration.

Information retrieved while loading subsequent `PROVISIONING_REFERENCE` URLs may replace previous key/values in the Provisioning Dictionary, but must not erase unrecognized key/values. For example, if an assignment has assigned the key `proprietary-x`, with a value '3', then later assignments must not override this value, unless the later loaded ZIP file contains an entry with that name. All these updates to the Provisioning Dictionary must be stored persistently. At the same time, each entry of type `bundle` or `bundle-url` (see Table 110.1) must be installed and not started.

Once the Management Agent has been started, the Initial Provisioning service has become operational. In this state, the Initial Provisioning service must react when the Provisioning Dictionary is updated with a new `PROVISIONING_REFERENCE` property. If this key is set, it should start the cycle again. For example, if the control of an OSGi framework needs to be transferred to another Remote Manager, the Management Agent should set the `PROVISIONING_REFERENCE` to the location of this new Remote Manager’s Initial Provisioning ZIP file. This process is called `re-provisioning`.

If errors occur during this process, the Initial Provisioning service should try to notify the Service User of the problem.

The previous description is depicted in Figure 110.2 as a flow chart.
The Management Agent may require configuration data that is specific to the OSGi framework instance. If this data is available outside the Management Agent bundle, the merging of this data with the Management Agent may take place in the OSGi framework. Transferring the data separately will make it possible to simplify the implementation on the server side, as it is not necessary to create personalized OSGi framework bundles. The **PROVISIONING_AGENT_CONFIG** key is reserved for this purpose, but the Management Agent may use another key or mechanisms if so desired.

The **PROVISIONING_SPID** key must contain the OSGi framework Identifier.

### 110.2.1 InitialProvisioning-Entries Manifest Header

The InitialProvisioning-Entries manifest header optionally specifies the type of the entries in the ZIP file. The syntax for this header is:

\[
\text{InitialProvisioning-Entries ::= ip-entry (',' ip-entry ) *} \\
\text{ip-entry ::= path ( ';' parameter ) *}
\]

The entry is the path name of a resource in the ZIP file. This InitialProvisioning-Entries header recognizes the following attribute:

- **type** - Gives the type of the dictionary entry. The type can have one of the following values: text, binary, bundle, or bundle-url

If the type parameter entry is not specified for an entry, then the type will be inferred from the extension of the entry, as defined in table Table 110.1 on page 151.

### 110.3 Special Configurations

The next section shows some examples of specially configured types of OSGi framework Servers and how they are treated with the respect to the specifications in this document.
**110.3.1 Branded OSGi framework Server**

If a OSGi framework Operator is selling OSGi framework Servers branded exclusively for use with their service, the provisioning will most likely be performed prior to shipping the OSGi framework Server to the User. Typically the OSGi framework is configured with the Dictionary entry `PROVISIONING_REFERENCE` pointing at a location controlled by the Operator.

Up-to-date bundles and additional configuration data must be loaded from that location at activation time. The OSGi framework is probably equipped with necessary security entities, like certificates, to enable secure downloads from the Operator’s URL over open networks, if necessary.

**110.3.2 Non-connected OSGi framework**

Circumstances might exist in which the OSGi framework Server has no WAN connectivity, or prefers not to depend on it for the purposes not covered by this specification.

The non-connected case can be implemented by specifying a file:// URL for the initial ZIP file (`PROVISIONING_REFERENCE`). That file:// URL would name a local file containing the response that would otherwise be received from a remote server.

The value for the Management Agent `PROVISIONING_REFERENCE` found in that file will be used as input to the load process. The `PROVISIONING_REFERENCE` may point to a bundle file stored either locally or remotely. No code changes are necessary for the non-connected scenario. The file:// URLs must be specified, and the appropriate files must be created on the OSGi framework.

**110.4 The Provisioning Service**

Provisioning information is conveyed between bundles using the Provisioning Service, as defined in the `ProvisioningService` interface. The Provisioning Dictionary is retrieved from the `ProvisioningService` object using the `getInformation()` method. This is a read-only Dictionary object, any changes to this Dictionary object must throw an `UnsupportedOperationException`.

The Provisioning Service provides a number of methods to update the Provisioning Dictionary.

- `addInformation(Dictionary)` - Add all key/value pairs in the given Dictionary object to the Provisioning Dictionary.
- `addInformation(ZipInputStream)` - It is also possible to add a ZIP file to the Provisioning Service immediately. This will unpack the ZIP file and add the entries to the Provisioning Dictionary. This method must install the bundles contained in the ZIP file as described in `Procedure` on page 150.
- `setInformation(Dictionary)` - Set a new Provisioning Dictionary. This will remove all existing entries.

Each of these method will increment the `PROVISIONING_UPDATE_COUNT` entry.

**110.5 Management Agent Environment**

The Management Agent should be written with great care to minimize dependencies on other packages and services, as all services in OSGi are optional. Some OSGi frameworks may have other bundles pre-installed, so it is possible that there may be exported packages and services available. Mechanisms outside the current specification, however, must be used to discover these packages and services before the Management Agent is installed.

The Provisioning Service must ensure that the Management Agent is running with `AllPermission`. The Management Agent should check to see if the Permission Admin service is available, and establish the initial permissions as soon as possible to insure the security of the device when later
bundles are installed. As the PermissionAdmin interfaces may not be present (it is an optional service), the Management Agent should export the PermissionAdmin interfaces to ensure they can be resolved.

Once started, the Management Agent may retrieve its configuration data from the Provisioning Service by getting the byte[] object that corresponds to the PROVISIONING_AGENT_CONFIG key in the Provisioning Dictionary. The structure of the configuration data is implementation specific.

The scope of this specification is to provide a mechanism to transmit the raw configuration data to the Management Agent. The Management Agent bundle may alternatively be packaged with its configuration data in the bundle, so it may not be necessary for the Management Agent bundle to use the Provisioning Service at all.

Most likely, the Management Agent bundle will install other bundles to provision the OSGi framework. Installing other bundles might even involve downloading a more full featured Management Agent to replace the initial Management Agent.

110.6 Mapping To File Scheme

The file: scheme is the simplest and most completely supported scheme which can be used by the Initial Provisioning specification. It can be used to store the configuration data and Management Agent bundle on the OSGi framework Server, and avoids any outside communication.

If the initial request URL has a file scheme, no parameters should be appended, because the file: scheme does not accept parameters.

110.6.1 Example With File Scheme

The manufacturer should prepare a ZIP file containing only one entry named PROVISIONING_START_BUNDLE that contains a location string of an entry of type bundle or bundle-url. For example, the following ZIP file demonstrates this:

```
provisioning.start.bundle text agent
agent bundle COAF0E9B2AB..
```

The bundle may also be specified with a URL:

```
provisioning.start.bundle text http://acme.com/a.jar
agent bundle-url http://acme.com/a.jar
```

Upon startup, the framework is provided with the URL with the file: scheme that points to this ZIP file:

```
file:/opt/osgi/ma.zip
```

110.7 Mapping To HTTP(S) Scheme

This section defines how HTTP and HTTPS URLs must be used with the Initial Provisioning specification.

- **HTTP** - May be used when the data exchange takes place over networks that are secured by other means, such as a Virtual Private Network (VPN) or a physically isolated network. Otherwise, HTTP is not a valid scheme because no authentication takes place.

- **HTTPS** - May be used if the OSGi framework is equipped with appropriate certificates.

HTTP and HTTPS share the following qualities:
• Both are well known and widely used
• Numerous implementations of the protocols exist
• Caching of the Management Agent will be desired in many implementations where limited bandwidth is an issue. Both HTTP and HTTPS already contain an accepted protocol for caching.

Both HTTP and HTTPS must be used with the GET method. The response is a ZIP file, implying that the response header Content-Type header must contain application/zip.

### 110.7 HTTPS Certificates

In order to use HTTPS, certificates must be in place. These certificates, that are used to establish trust towards the Operator, may be made available to the OSGi framework using the Provisioning Service. The root certificate should be assigned to the Provisioning Dictionary before the HTTPS provider is used. Additionally, the OSGi framework should be equipped with a OSGi framework certificate that allows the OSGi framework to properly authenticate itself towards the Operator. This specification does not state how this certificate gets installed into the OSGi framework.

The root certificate is stored in the Provisioning Dictionary under the key:

```
PROVISIONING_ROOTX509
```

The Root X.509 Certificate holds certificates used to represent a handle to a common base for establishing trust. The certificates are typically used when authenticating a Remote Manager to the OSGi framework. In this case, a Root X.509 certificate must be part of a certificate chain for the Operator’s certificate. The format of the certificate is defined in [Certificate Encoding](#) on page 156.

### 110.7.2 Certificate Encoding

Root certificates are X.509 certificates. Each individual certificate is stored as a byte[] object. This byte[] object is encoded in the default Java manner, as follows:

- The original, binary certificate data is DER encoded
- The DER encoded data is encoded into base64 to make it text.
- The base64 encoded data is prefixed with:

  ```
  -----BEGIN CERTIFICATE-----
  ```

- The decoding of such a certificate may be done with the java.security.cert.CertificateFactory class:

  ```java
  InputStream bis = new ByteArrayInputStream(x509);// byte[]
  CertificateFactory cf =
    CertificateFactory.getInstance("X.509");
  Collection c = cf.generateCertificates(bis);
  Iterator i = c.iterator();
  while (i.hasNext()) {
    Certificate cert = (Certificate)i.next();
    System.out.println(cert);
  }
  ```
110.7.3 URL Encoding

The URL must contain the OSGi framework Identity, and may contain more parameters. These parameters are encoded in the URL according to the HTTP(S) URL scheme. A base URL may be set by an end user but the Provisioning Service must add the OSGi framework Identifier.

If the request URL already contains HTTP parameters (if there is a '?' in the request), the service_platform_id is appended to this URL as an additional parameter. If, on the other hand, the request URL does not contain any HTTP parameters, the service_platform_id will be appended to the URL after a '?', becoming the first HTTP parameter. The following two examples show these two variants:

http://server.operator.com/service-x? foo=bar&service_platform_id=VIN:123456789

http://server.operator.com/service-x? service_platform_id=VIN:123456789

Proper URL encoding must be applied when the URL contains characters that are not allowed. See [6] RFC 2396 - Uniform Resource Identifier (URI).

110.8 Mapping To RSH Scheme

The RSH protocol is an OSGi-specific protocol, and is included in this specification because it is optimized for Initial Provisioning. It requires a shared secret between the management system and the OSGi framework that is small enough to be entered by the Service User.

RSH bases authentication and encryption on Message Authentication Codes (MACs) that have been derived from a secret that is shared between the OSGi framework and the Operator prior to the start of the protocol execution.

The protocol is based on an ordinary HTTP GET request/response, in which the request must be signed and the response must be encrypted and authenticated. Both the signature and encryption key are derived from the shared secret using Hashed Message Access Codes (HMAC) functions.

As additional input to the HMAC calculations, one client-generated nonce and one server-generated nonce are used to prevent replay attacks. The nonces are fairly large random numbers that must be generated in relation to each invocation of the protocol, in order to guarantee freshness. These nonces are called clientfg (client-generated freshness guarantee) and serverfg (server-generated freshness guarantee).

In order to separate the HMAC calculations for authentication and encryption, each is based on a different constant value. These constants are called the authentication constant and the encryption constant.

From an abstract perspective, the protocol may be described as follows.

- δ - Shared secret, 160 bits or more
- s - Server nonce, called servercfg, 128 bits
- c - Client nonce, called clientfg, 128 bits
- K_a - Authentication key, 160 bits
- K_e - Encryption key, 192 bits
- r - Response data
- e - Encrypted data
- E - Encryption constant, a byte[] of 05, 36, 54, 70, 00 (hex)
- A - Authentication constant, a byte[] of 00, 4f, 53, 47, 49 (hex)
• $M$ - Message material, used for $K_e$ calculation.
• $m$ - The calculated message authentication code.
• $3DES$ - Triple DES, encryption function, see [8] $3DES$. The bytes of the key must be set to odd parity. CBC mode must be used where the padding method is defined in [9] RFC 1423 Part III: Algorithms, Modes, and Identifiers. In [11] Java Cryptography API (part of Java 1.4) this is addressed as PKCS5Padding.
• $IV$ - Initialization vector for 3DES.
• $SHA1$ - Secure Hash Algorithm to generate the Hashed Message Authentication Code, see [12] SHA-1. The function takes a single parameter, the block to be worked upon.
• $HMAC$ - The function that calculates a message authentication code, which must HMAC-SHA1. HMAC-SHA1 is defined in [1] HMAC: Keyed-Hashing for Message Authentication. The HMAC function takes a key and a block to be worked upon as arguments. Note that the lower 16 bytes of the result must be used.
• $\{\}$ - Concatenates its arguments
• $[\]$ - Indicates access to a sub-part of a variable, in bytes. Index starts at one, not zero.

In each step, the emphasized server or client indicates the context of the calculation. If both are used at the same time, each variable will have server or client as a subscript.

1. The client generates a random nonce, stores it and denotes it $clientfg$
   \[ c = nonce \]
2. The client sends the request with the $clientfg$ to the server.
   \[ c_{server} \leftarrow c_{client} \]
3. The server generates a nonce and denotes it $serverfg$.
   \[ s = nonce \]
4. The server calculates an authentication key based on the SHA1 function, the shared secret, the received $clientfg$, the $serverfg$ and the authentication constant.
   \[ K_s \leftarrow SHA1(\{\delta, c, s, A\}) \]
5. The server calculates an encryption key using an SHA-1 function, the shared secret, the received $clientfg$, the $serverfg$ and the encryption constant. It must first calculate the key material $M$.
   \[ M[1, 20] \leftarrow SHA1(\{\delta, c, s, E\}) \]
   \[ M[21, 40] \leftarrow SHA1(\{\delta, M[1, 20], c, s, E\}) \]
6. The key for DES consists $K_e$ and $IV$.
   \[ K_e \leftarrow M[1, 24] \]
   \[ IV \leftarrow M[25, 32] \]

   The server encrypts the response data using the encryption key derived in step 5. The encryption algorithm that must be used to encrypt/decrypt the response data is 3DES. 24 bytes (192 bits) from $M$ are used to generate $K_e$, but the low order bit of each byte must be used as an odd parity bit. This means that before using $K_e$, each byte must be processed to set the low order bit so that the byte has odd parity.

   The encryption/decryption key used is specified by the following:
   \[ e \leftarrow 3DES(K_e, IV, r) \]
7. The server calculates a MAC $m$ using the HMAC function, the encrypted response data and the authentication key derived in 4.
   \[ m \leftarrow HMAC(\delta, e) \]
8. The server sends a response to the client containing the $serverfg$, the MAC $m$ and the encrypted response data
The client calculates the encryption key $K_e$ the same way the server did in steps 5 and 6, and uses this to decrypt the encrypted response data. The serverfg value received in the response is used in the calculation.

$$r \leftarrow 3DES(K_e, IV, e)$$

9. The client performs the calculation of the MAC $m'$ in the same way the server did, and checks that the results match the received MAC $m$. If they do not match, further processing is discarded. The serverfg value received in the response is used in the calculation.

$$K_a \leftarrow \text{SHA1}([\delta, c, s, A])$$

$$m' \leftarrow \text{HMAC}(K_a, e)$$

$$m' = m$$

**Figure 110.3** Action Diagram for RSH

The *shared secret* should be a key of length 160 bits (20 bytes) or more. The length is selected to match the output of the selected hash algorithm [2] *NIST, FIPS PUB 180-1: Secure Hash Standard*, April 1995.

In some scenarios, the shared secret is generated by the Operator and communicated to the User, who inserts the secret into the OSGi framework through some unspecified means.

The opposite is also possible: the shared secret can be stored within the OSGi framework, extracted from it, and then communicated to the Operator. In this scenario, the source of the shared secret could be either the OSGi framework or the Operator.

In order for the server to calculate the authentication and encryption keys, it requires the proper shared secret. The server must have access to many different shared secrets, one for each OSGi framework it is to support. To be able to resolve this issue, the server must typically also have access to the OSGi framework Identifier of the OSGi framework. The normal way for the server to know the OSGi framework Identifier is through the application protocol, as this value is part of the URL encoded parameters of the HTTP, HTTPS, or RSH mapping of the Initial Provisioning.

In order to be able to switch Operators, a new shared secret must be used. The new secret may be generated by the new Operator and then inserted into the OSGi framework device using a mechanism not covered by this specification. Or the device itself may generate the new secret and convey it to the owner of the device using a display device or read-out, which is then communicated to the new operator out-of-band. Additionally, the generation of the new secret may be triggered by some external event, like holding down a button for a specified amount of time.
110.8.2 Request Coding

RSH is mapped to HTTP or HTTPS. Thus, the request parameters are URL encoded as discussed in URL Encoding on page 157. RSH requires an additional parameter in the URL: the clientfg parameter. This parameter is a nonce that is used to counter replay attacks. See also RSH Transport on page 160.

110.8.3 Response Coding

The server’s response to the client is composed of three parts:

- A header containing the protocol version and the serverfg
- The MAC
- The encrypted response

These three items are packaged into a binary container according to Table 110.2.

Table 110.2 RSH Header description

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
<th>Value hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Number of bytes in header</td>
<td>2E</td>
</tr>
<tr>
<td>1</td>
<td>Major version number</td>
<td>01</td>
</tr>
<tr>
<td>1</td>
<td>Minor version number</td>
<td>00</td>
</tr>
<tr>
<td>16</td>
<td>serverfg</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>Number of bytes in MAC</td>
<td>10</td>
</tr>
<tr>
<td>16</td>
<td>Message Authentication Code</td>
<td>MAC</td>
</tr>
<tr>
<td>4</td>
<td>Number of bytes of encrypted ZIP file</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>Encrypted ZIP file</td>
<td>...</td>
</tr>
</tbody>
</table>

The response content type is an RSH-specific encrypted ZIP file, implying that the response header Content-Type must be application/x-rsh for the HTTP request. When the content file is decrypted, the content must be a ZIP file.

110.8.4 RSH URL

The RSH URL must be used internally within the OSGi framework to indicate the usage of RSH for initial provisioning. The RSH URL format is identical to the HTTP URL format, except that the scheme is rsh: instead of http: For example (« means line continues on next line):

rsh://server.operator.com/service-x

110.8.5 Extensions to the Provisioning Service Dictionary

RSH specifies one additional entry for the Provisioning Dictionary:

PROVISIONING_RSH_SECRET

The value of this entry is a byte[] containing the shared secret used by the RSH protocol.

110.8.6 RSH Transport

RSH is mapped to HTTP or HTTPS and follows the same URL encoding rules, except that the clientfg parameter is additionally appended to the URL. The key in the URL must be clientfg and the value must be encoded in base 64 format:

The clientfg parameter is transported as an HTTP parameter that is appended after the service_platform_id parameter. The second example above would then be:

rsh://server.operator.com/service-x
Which, when mapped to HTTP, must become:

```
http://server.operator.com/service-x «
  service_platform_id=VIN:123456789& «
  clientfg=AHPmWcw%2FsiWYC37xZNdkvQ%23D%23D
```

## 110.9 Exception Handling

The Initial Provisioning process is a sensitive process that must run without user supervision. There is therefore a need to handle exceptional cases in a well-defined way to simplify troubleshooting.

There are only 2 types of problems that halt the provisioning process. They are:

- IO Exception when reading or writing provisioning information.
- IO Exception when retrieving or processing a provisioning zip file.

Other exceptions can occur and the Provisioning Service must do any attempt to log these events. In the cases that the provisioning process stops, it is important that the clients of the provisioning service have a way to find out that the process is stopped. The mechanism that is used for this is a special entry in the provisioning dictionary. The name of the entry must be `provisioning.error`. The value is a String object with the following format:

- Numeric error code
- Space
- A human-readable string describing the error.

Permitted error codes are:

- 0 - Unknown error
- 1 - Couldn't load or save provisioning information
- 2 - Malformed URL Exception
- 3 - IO Exception when retrieving document of a URL
- 4 - Corrupted Zip Input Stream

The provisioning.update.count will be incremented as normal when a `provisioning.error` entry is added to the provisioning information. After, the provisioning service will take no further action.

Some examples:

```
0 SIM card removed
2 "http://www.acme.com/secure/blib/ifa.zip"
```

## 110.10 Security

The security model for the OSGi framework is based on the integrity of the Management Agent deployment. If any of the mechanisms used during the deployment of management agents are weak, or can be compromised, the whole security model becomes weak.

From a security perspective, one attractive means of information exchange would be a smart card. This approach enables all relevant information to be stored in a single place. The Operator could then provide the information to the OSGi framework by inserting the smart card into the OSGi framework.
110.10.1 Concerns

The major security concerns related to the deployment of the Management Agent are:

- The OSGi framework is controlled by the intended Operator
- The Operator controls the intended OSGi framework(s)
- The integrity and confidentiality of the information exchange that takes place during these processes must be considered

In order to address these concerns, an implementation of the OSGi Remote Management Architecture must assure that:

- The Operator authenticates itself to the OSGi framework
- The OSGi framework authenticates itself to the Operator
- The integrity and confidentiality of the Management Agent, certificates, and configuration data are fully protected if they are transported over public transports.

Each mapping of the Initial Provisioning specification to a concrete implementation must describe how these goals are met.

110.10.2 OSGi framework Long-Term Security

Secrets for long-term use may be exchanged during the Initial Provisioning procedures. This way, one or more secrets may be shared securely, assuming that the Provisioning Dictionary assignments used are implemented with the proper security characteristics.

110.10.3 Permissions

The provisioning information may contain sensitive information. Also, the ability to modify provisioning information can have drastic consequences. Thus, only trusted bundles should be allowed to register, or get the Provisioning Service. This restriction can be enforced using ServicePermission[ ProvisioningService, GET].

No Permission classes guard reading or modification of the Provisioning Dictionary, so care must be taken not to leak the Dictionary object received from the Provisioning Service to bundles that are not trusted.

Whether message-based or connection-based, the communications used for Initial Provisioning must support mutual authentication and message integrity checking, at a minimum.

By using both server and client authentication in HTTPS, the problem of establishing identity is solved. In addition, HTTPS will encrypt the transmitted data. HTTPS requires a Public Key Infrastructure implementation in order to retrieve the required certificates.

When RSH is used, it is vital that the shared secret is shared only between the Operator and the OSGi framework, and no one else.

110.11 org.osgi.service.provisioning

Provisioning Package Version 1.2.

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.provisioning; version="[1.2,2.0)"
Example import for providers implementing the API in this package:

```java
Import-Package: org.osgi.service.provisioning; version="[1.2,1.3)"
```

### 110.11.1 public interface ProvisioningService

Service for managing the initial provisioning information.

Initial provisioning of an OSGi device is a multi step process that culminates with the installation and execution of the initial management agent. At each step of the process, information is collected for the next step. Multiple bundles may be involved and this service provides a means for these bundles to exchange information. It also provides a means for the initial Management Bundle to get its initial configuration information.

The provisioning information is collected in a `Dictionary` object, called the Provisioning Dictionary. Any bundle that can access the service can get a reference to this object and read and update provisioning information. The key of the dictionary is a `String` object and the value is a `String` or `byte[]` object. The single exception is the `PROVISIONING_UPDATE_COUNT` value which is an `Integer`. The provisioning prefix is reserved for keys defined by OSGi, other key names may be used for implementation dependent provisioning systems.

Any changes to the provisioning information will be reflected immediately in all the dictionary objects obtained from the Provisioning Service.

Because of the specific application of the Provisioning Service, there should be only one Provisioning Service registered. This restriction will not be enforced by the Framework. Gateway operators or manufacturers should ensure that a Provisioning Service bundle is not installed on a device that already has a bundle providing the Provisioning Service.

The provisioning information has the potential to contain sensitive information. Also, the ability to modify provisioning information can have drastic consequences. Thus, only trusted bundles should be allowed to register and get the Provisioning Service. The `ServicePermission` is used to limit the bundles that can gain access to the Provisioning Service. There is no check of `Permission` objects to read or modify the provisioning information, so care must be taken not to leak the Provisioning Dictionary received from `getInformation` method.

**No Implement** Consumers of this API must not implement this interface

#### 110.11.1.1 public static final String INITIALPROVISIONING_ENTRIES = "InitialProvisioning-Entries"

Name of the header that specifies the type information for the ZIP file entries.

*Since* 1.2

#### 110.11.1.2 public static final String MIME_BUNDLE = "application/vnd.osgi.bundle"

MIME type to be stored in the extra field of a `ZipEntry` object for an installable bundle file. Zip entries of this type will be installed in the framework, but not started. The entry will also not be put into the information dictionary.

*Since* 1.2

#### 110.11.1.3 public static final String MIME_BUNDLE_ALT = "application/x-osgi-bundle"

Alternative MIME type to be stored in the extra field of a `ZipEntry` object for an installable bundle file. Zip entries of this type will be installed in the framework, but not started. The entry will also not be put into the information dictionary. This alternative entry is only for backward compatibility, new applications are recommended to use `MIME_BUNDLE`, which is an official IANA MIME type.

*Since* 1.2

#### 110.11.1.4 public static final String MIME_BUNDLE_URL = "text/x-osgi-bundle-url"

MIME type to be stored in the extra field of a `ZipEntry` for a String that represents a URL for a bundle. Zip entries of this type will be used to install (but not start) a bundle from the URL. The entry will not be put into the information dictionary.
110.11.1.5 **public static final String MIME_BYTE_ARRAY = "application/octet-stream"**

MIME type to be stored stored in the extra field of a ZipEntry object for byte[] data.

110.11.1.6 **public static final String MIME_STRING = "text/plain; charset=utf-8"**

MIME type to be stored in the extra field of a ZipEntry object for String data.

110.11.1.7 **public static final String PROVISIONING_AGENT_CONFIG = "provisioning.agent.config"**

The key to the provisioning information that contains the initial configuration information of the initial Management Agent. The value will be of type byte[].

110.11.1.8 **public static final String PROVISIONING_REFERENCE = "provisioning.reference"**

The key to the provisioning information that contains the location of the provision data provider. The value must be of type String.

110.11.1.9 **public static final String PROVISIONING_ROOTX509 = "provisioning.rootx509"**

The key to the provisioning information that contains the root X509 certificate used to establish trust with operator when using HTTPS.

110.11.1.10 **public static final String PROVISIONING_RSH_SECRET = "provisioning.rsh.secret"**

The key to the provisioning information that contains the shared secret used in conjunction with the RSH protocol.

110.11.1.11 **public static final String PROVISIONING_SPID = "provisioning.spid"**

The key to the provisioning information that uniquely identifies the Service Platform. The value must be of type String.

110.11.1.12 **public static final String PROVISIONING_START_BUNDLE = "provisioning.start.bundle"**

The key to the provisioning information that contains the location of the bundle to start with AllPermission. The bundle must have be previously installed for this entry to have any effect.

110.11.1.13 **public static final String PROVISIONING_UPDATE_COUNT = "provisioning.update.count"**

The key to the provisioning information that contains the update count of the info data. Each set of changes to the provisioning information must end with this value being incremented. The value must be of type Integer. This key/value pair is also reflected in the properties of the ProvisioningService in the service registry.

110.11.1.14 **public void addInformation(Dictionary info)**

- info the set of Provisioning Information key/value pairs to add to the Provisioning Information dictionary. Any keys are values that are of an invalid type will be silently ignored.

- □ Adds the key/value pairs contained in info to the Provisioning Information dictionary. This method causes the PROVISIONING_UPDATE_COUNT to be incremented.

110.11.1.15 **public void addInformation(ZipInputStream zis) throws IOException**

- zis the ZipInputStream that will be used to add key/value pairs to the Provisioning Information dictionary and install and start bundles. If a ZipEntry does not have an Extra field that corresponds to one of the four defined MIME types (MIME_STRING, MIME_BYTE_ARRAY, MIME_BUNDLE, and MIME_BUNDLE_URL) in will be silently ignored.

- □ Processes the ZipInputStream and extracts information to add to the Provisioning Information dictionary, as well as, install/update and start bundles. This method causes the PROVISIONING_UPDATE_COUNT to be incremented.
Throws IOException – if an error occurs while processing the ZipInputStream. No additions will be made to the Provisioning Information dictionary and no bundles must be started or installed.

110.11.16 public Dictionary getInformation()

- Returns a reference to the Provisioning Dictionary. Any change operations (put and remove) to the dictionary will cause an UnsupportedOperationException to be thrown. Changes must be done using the setInformation and addInformation methods of this service.

Returns A reference to the Provisioning Dictionary.

110.11.17 public void setInformation(Dictionary info)

- info the new set of Provisioning Information key/value pairs. Any keys are values that are of an invalid type will be silently ignored.

- Replaces the Provisioning Information dictionary with the key/value pairs contained in info. Any key/value pairs not in info will be removed from the Provisioning Information dictionary. This method causes the PROVISIONING_UPDATE_COUNT to be incremented.

110.12 References


[3] Hypertext Transfer Protocol - HTTP/1.1


[5] ZIP Archive format
http://www.pkware.com/support/zip-app-note/archives

[6] RFC 2396 - Uniform Resource Identifier (URI)
http://www.ietf.org/rfc/rfc2396.txt

[7] MIME Types
http://www.ietf.org/rfc/rfc2046.txt
http://www.iana.org/assignments/media-types

[8] 3DES

http://www.ietf.org/rfc/rfc1423.txt

[10] PKCS 5


[12] SHA-1

112 Declarative Services Specification

Version 1.2

112.1 Introduction

The OSGi Framework contains a procedural service model which provides a publish/find/bind model for using services. This model is elegant and powerful, it enables the building of applications out of bundles that communicate and collaborate using these services.

This specification addresses some of the complications that arise when the OSGi service model is used for larger systems and wider deployments, such as:

- **Startup Time** - The procedural service model requires a bundle to actively register and acquire its services. This is normally done at startup time, requiring all present bundles to be initialized with a Bundle Activator. In larger systems, this quickly results in unacceptably long startup times.

- **Memory Footprint** - A service registered with the Framework implies that the implementation, and related classes and objects, are loaded in memory. If the service is never used, this memory is unnecessarily occupied. The creation of a class loader may therefore cause significant overhead.

- **Complexity** - Service can come and go at any time. This dynamic behavior makes the service programming model more complex than more traditional models. This complexity negatively influences the adoption of the OSGi service model as well as the robustness and reliability of applications because these applications do not always handle the dynamicity correctly.

The service component model uses a declarative model for publishing, finding and binding to OSGi services. This model simplifies the task of authoring OSGi services by performing the work of registering the service and handling service dependencies. This minimizes the amount of code a programmer has to write; it also allows service components to be loaded only when they are needed. As a result, bundles need not provide a BundleActivator class to collaborate with others through the service registry.

From a system perspective, the service component model means reduced startup time and potentially a reduction of the memory footprint. From a programmer's point of view the service component model provides a simplified programming model.

The Service Component model makes use of concepts described in [1] Automating Service Dependency Management in a Service-Oriented Component Model.

112.1.1 Essentials

- **Backward Compatibility** - The service component model must operate seamlessly with the existing service model.

- **Size Constraints** - The service component model must not require memory and performance intensive subsystems. The model must also be applicable on resource constrained devices.

- **Delayed Activation** - The service component model must allow delayed activation of a service component. Delayed activation allows for delayed class loading and object creation until needed, thereby reducing the overall memory footprint.

- **Simplicity** - The programming model for using declarative services must be very simple and not require the programmer to learn a complicated API or XML sub-language.
• **Reactive** - It must be possible to react to changes in the external dependencies with different policies.

• **Annotations** - Annotations must be provided that can leverage the type information to create the XML descriptor.

### 112.1.2 Entities

- **Service Component** - A service component contains a description that is interpreted at run time to create and dispose objects depending on the availability of other services, the need for such an object, and available configuration data. Such objects can optionally provide a service. This specification also uses the generic term *component* to refer to a service component.

- **Component Description** - The declaration of a service component. It is contained within an XML document in a bundle.

- **Component Properties** - A set of properties which can be specified by the component description, Configuration Admin service and from the component factory.

- **Component Configuration** - A component configuration represents a component description parameterized by component properties. It is the entity that tracks the component dependencies and manages a component instance. An activated component configuration has a component context.

- **Component Instance** - An instance of the component implementation class. A component instance is created when a component configuration is activated and discarded when the component configuration is deactivated. A component instance is associated with exactly one component configuration.

- **Delayed Component** - A component whose component configurations are activated when their service is requested.

- **Immediate Component** - A component whose component configurations are activated immediately upon becoming satisfied.

- **Factory Component** - A component whose component configurations are created and activated through the component's component factory.

- **Reference** - A specified dependency of a component on a set of target services.

- **Service Component Runtime (SCR)** - The actor that manages the components and their life cycle.

- **Target Services** - The set of services that is defined by the reference interface and target property filter.

- **Bound Services** - The set of target services that are bound to a component configuration.

- **Event methods** - The bind, updated, and unbind methods associated with a Reference.
112.1.3 Synopsis

The Service Component Runtime reads component descriptions from started bundles. These descriptions are in the form of XML documents which define a set of components for a bundle. A component can refer to a number of services that must be available before a component configuration becomes satisfied. These dependencies are defined in the descriptions and the specific target services can be influenced by configuration information in the Configuration Admin service. After a component configuration becomes satisfied, a number of different scenarios can take place depending on the component type:

- **Immediate Component** - The component configuration of an immediate component must be activated immediately after becoming satisfied. Immediate components may provide a service.
- **Delayed Component** - When a component configuration of a delayed component becomes satisfied, SCR will register the service specified by the service element without activating the component configuration. If this service is requested, SCR must activate the component configuration creating an instance of the component implementation class that will be returned as the service object. If the servicefactory attribute of the service element is true, then, for each distinct bundle that requests the service, a different component configuration is created and activated and a new instance of the component implementation class is returned as the service object.
- **Factory Component** - If a component's description specifies the factory attribute of the component element, SCR will register a Component Factory service. This service allows client bundles to create and activate multiple component configurations and dispose of them. If the component's description also specifies a service element, then as each component configuration is activated, SCR will register it as a service.

112.1.4 Readers

- **Architects** - The chapter, Components on page 170, gives a comprehensive introduction to the capabilities of the component model. It explains the model with a number of examples. The section about Component Life Cycle on page 187 provides some deeper insight in the life cycle of components.
- **Service Programmers** - Service programmers should read Components on page 170. This chapter should suffice for the most common cases. For the more advanced possibilities, they should con-
Components

A component is a normal Java class contained within a bundle. The distinguishing aspect of a component is that it is declared in an XML document. Component configurations are activated and deactivated under the full control of SCR. SCR bases its decisions on the information in the component's description. This information consists of basic component information like the name and type, optional services that are implemented by the component, and references. References are dependencies that the component has on other services.

SCR must activate a component configuration when the component is enabled and the component configuration is satisfied and a component configuration is needed. During the life time of a component configuration, SCR can notify the component of changes in its bound references.

SCR will deactivate a previously activated component configuration when the component becomes disabled, the component configuration becomes unsatisfied, or the component configuration is no longer needed.

If an activated component configuration's configuration properties change, SCR must deactivate the component configuration and then attempt to reactivate the component configuration using the new configuration information.

112.2.1 Declaring a Component

A component requires the following artifacts in the bundle:

- An XML document that contains the component description.
- The Service-Component manifest header which names the XML documents that contain the component descriptions.
- An implementation class that is specified in the component description.

The elements in the component's description are defined in Component Description on page 179. The XML grammar for the component declaration is defined by the XML Schema, see Component Description Schema on page 202.

Immediate Component

An immediate component is activated as soon as its dependencies are satisfied. If an immediate component has no dependencies, it is activated immediately. A component is an immediate component if it is not a factory component and either does not specify a service or specifies a service and the immediate attribute of the component element set to true. If an immediate component configuration is satisfied and specifies a service, SCR must register the component configuration as a service in the service registry and then activate the component configuration.

For example, the bundle entry /OSGI-INF/activator.xml contains:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scr:component name="example.activator"
    xmlns:scr="http://www.osgi.org/xmlns/scr/v1.2.0">
    <implementation class="com.acme.Activator"/>
</scr:component>
```

The manifest header Service-Component must also be specified in the bundle manifest. For example:
112.2.3 Delayed Component

A delayed component specifies a service, is not specified to be a factory component and does not have the immediate attribute of the component element set to true. If a delayed component configuration is satisfied, SCR must register the component configuration as a service in the service registry but the activation of the component configuration is delayed until the registered service is requested. The registered service of a delayed component looks like a normal registered service but does not incur the overhead of an ordinarily registered service that require a service’s bundle to be initialized to register the service.

For example, a bundle needs to see events of a specific topic. The Event Admin uses the white board pattern, receiving the events is therefore as simple as registering a Event Handler service. The example XML for the delayed component looks like:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scr:component name="example.handler"
xmlns:scr="http://www.osgi.org/xmlns/scr/v1.2.0">
  <implementation class="com.acme.HandlerImpl"/>
  <property name="event.topics">some/topic</property>
  <service>
    <provide interface="org.osgi.service.event.EventHandler"/>
  </service>
</scr:component>
```

The associated component class looks like:

```java
public class HandlerImpl implements EventHandler{
    public void handleEvent(Event evt ) {
      ...
    }
}
```

The component configuration will only be activated once the Event Admin service requires the service because it has an event to deliver on the topic to which the component subscribed.

112.2.4 Factory Component

Certain software patterns require the creation of component configurations on demand. For example, a component could represent an application that can be launched multiple times and each ap-
Application instance can then quit independently. Such a pattern requires a factory that creates the instances. This pattern is supported with a factory component. A factory component is used if the factory attribute of the component element is set to a factory identifier. This identifier can be used by a bundle to associate the factory with externally defined information.

SCR must register a Component Factory service on behalf of the component as soon as the component factory is satisfied. The service properties must be:

- component.name - The name of the component.
- component.factory - The factory identifier.

The service properties of the Component Factory service must not include the component properties.

New configurations of the component can be created and activated by calling the newInstance method on this Component Factory service. The newInstance(Dictionary) method has a Dictionary object as argument. This Dictionary object is merged with the component properties as described in Component Properties on page 196. If the component specifies a service, then the service is registered after the created component configuration is satisfied with the component properties. Then the component configuration is activated.

For example, a component can provide a connection to a USB device. Such a connection should normally not be shared and should be created each time such a service is needed. The component description to implement this pattern looks like:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scr:component name="example.factory"
    factory="usb.connection"
    xmlns:scr="http://www.osgi.org/xmlns/scr/v1.2.0">
    <implementation class="com.acme.USBConnectionImpl"/>
</scr:component>
```

The component class looks like:

```java
public class USBConnectionImpl implements USBConnection {
    private void activate(Map properties) {
        ...
    }
}
```

A factory component can be associated with a service. In that case, such a service is registered for each component configuration. For example, the previous example could provide a USB Connection service.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scr:component name="example.factory"
    factory="usb.connection"
    xmlns:scr="http://www.osgi.org/xmlns/scr/v1.2.0">
    <implementation class="com.acme.USBConnectionImpl"/>
    <service>
        <provide interface="com.acme.USBCnection"/>
    </service>
</scr:component>
```

The associated component class looks like:

```java
public class USBConnectionImpl implements USBCconnection {
    private void activate(Map properties) {
        ...
    }
}
public void connect() { ... }
...
public void close() { ... }
}

A new service will be registered each time a new component configuration is created and activated with the newInstance method. This allows a bundle other than the one creating the component configuration to utilize the service. If the component configuration is deactivated, the service must be unregistered.

112.3 References to Services

Most bundles will require access to other services from the service registry. The dynamics of the service registry require care and attention of the programmer because referenced services, once acquired, could be unregistered at any moment. The component model simplifies the handling of these service dependencies significantly.

The services that are selected by a reference are called the target services. These are the services selected by the BundleContext.getServiceReferences method where the first argument is the reference's interface and the second argument is the reference's target property, which must be a valid filter.

A component configuration becomes satisfied when each specified reference is satisfied. A reference is satisfied if it specifies optional cardinality or when the target services contains at least one member. An activated component configuration that becomes unsatisfied must be deactivated.

During the activation of a component configuration, SCR must bind some or all of the target services of a reference to the component configuration. Any target service that is bound to the component configuration is called a bound service. See Binding Services on page 191.

112.3.1 Accessing Services

A component instance must be able to use the services that are referenced by the component configuration, that is, the bound services of the references. There are two strategies for a component instance to acquire these bound services:

- **Event strategy** - SCR calls a method on the component instance when a service becomes bound, when a service becomes unbound, or when its properties are updated. These methods are the bind, updated, and unbind methods specified by the reference. The event strategy is useful if the component needs to be notified of changes to the bound services for a dynamic reference.
- **Lookup strategy** - A component instance can use one of the locateService methods of ComponentContext to locate a bound service. These methods take the name of the reference as a parameter. If the reference has a dynamic policy, it is important to not store the returned service object(s) but look it up every time it is needed.

A component may use either or both strategies to access bound services.

112.3.2 Event Methods

When using the event strategy the SCR must callback the components at the appropriate time. The SCR must callback on the following events:

- **bind** - The bind method is called to bind a new service to the component that matches the selection criteria. If the policy is dynamic then the bind method of a replacement service can be called before its corresponding unbind method.
- **updated** - The updated method is called when the service properties of a bound services are modified and the resulting properties do not cause the service to become unbound because it is no longer selected by the target filter.
References to Services

- **unbind** - The unbind method is called when the SCR needs to unbind the service.

Together these methods are called the *event methods*. Event methods must have one of the following prototypes:

```java
void <method-name>(ServiceReference);
void <method-name>(<parameter-type>);
void <method-name>(<parameter-type>, Map);
```

If an event method has the first prototype, then a Service Reference to the bound service will be passed to the method. This Service Reference may later be passed to the `locateService(String, ServiceReference)` method to obtain the actual service object. This approach is useful when the service properties need to be examined before accessing the service object. It also allows for the delayed activation of bound services when using the event strategy.

If an event method has the second prototype, then the service object of the bound service is passed to the method. The method's parameter type must be assignable from the type specified by the reference's interface attribute. That is, the service object of the bound service must be castable to the method's parameter type.

If an event method has the third prototype, then the service object of the bound service is passed to the method as the first argument and an unmodifiable Map containing the service properties of the bound service is passed as the second argument. The method's first parameter type must be assignable from the type specified by the reference's interface attribute. That is, the service object of the bound service must be castable to the method's parameter type.

The bind and unbind methods must be called once for each bound service. This implies that if the reference has multiple cardinality, then the methods may be called multiple times. The updated method can be called multiple times per service.

A suitable method is selected using the following priority:

1. The method takes a single argument and the type of the argument is `org.osgi.framework.ServiceReference`.
2. The method takes a single argument and the type of the argument is the type specified by the reference's interface attribute.
3. The method takes a single argument and the type of the argument is assignable from the type specified by the reference's interface attribute. If multiple methods match this rule, this implies the method name is overloaded and SCR may choose any of the methods to call.
4. The method takes two argument and the type of the first argument is specified by the reference's interface attribute and the type of the second argument is `java.util.Map`. If multiple methods match this rule, this implies the method name is overloaded and SCR may choose any of the methods to call.
5. The method takes two argument and the type of the first argument is assignable from the type specified by the reference's interface attribute and the type of the second argument is `java.util.Map`. If multiple methods match this rule, this implies the method name is overloaded and SCR may choose any of the methods to call.

When searching for an event method to call, SCR must locate a suitable method as specified in *Locating Component Methods* on page 201. If no suitable method is located, SCR must log an error message with the Log Service, if present, and there will be no bind, updated, or unbind notification.

When the service object for a bound service is first provided to a component instance, that is passed to an event method or returned by a locate service method, SCR must get the service object from the OSGi Framework's service registry using the `getService` method on the component's Bundle Context. If the service object for a bound service has been obtained and the service becomes unbound, SCR must unget the service object using the `ungetService` method on the component's Bundle Context and discard all references to the service object.

For example, a component requires the Log Service and uses the lookup strategy. The reference is declared without any bind, updated, and unbind methods:
The component implementation class must now lookup the service. This looks like:

```java
public class LogLookupImpl {
    private void activate(ComponentContext ctxt) {
        LogService log = (LogService)
                ctxt.locateService("LOG");
        log.log(LogService.LOG_INFO, "Hello Components!");
    }
}
```

Alternatively, the component could use the event strategy and ask to be notified with the Log Service by declaring bind, updated, and unbind methods.

```xml
<scr:component name="example.listen"
    xmlns:scr="http://www.osgi.org/xmlns/scr/v1.2.0">
    <implementation class="com.acme.LogEventImpl"/>
    <reference name="LOG" interface="org.osgi.service.log.LogService"
        bind="setLog"
        updated="updatedLog"
        unbind="unsetLog"/>
</scr:component>
```

The component implementation class looks like:

```java
public class LogEventImpl {
    LogService log;
    Integer level;
    void setLog( LogService l, Map<String,?> ref ) {
        log = l;
        updatedLog(ref);
    }
    void updatedLog( LogService l, Map<String,?> ref) {
        level = (Integer) ref.get("level");
    }
    void unsetLog( LogService l ) { log = null; }
    private void activate() {
        log.log(LogService.LOG_INFO, "Hello Components!");
    }
}
```

Event methods can be declared private in the component class but are only looked up in the inheritance chain when they are protected, public, or have default access. See *Locating Component Methods* on page 201.
112.3.3 Reference Cardinality

A component implementation is always written with a certain cardinality in mind. The cardinality represents two important concepts:

- **Multiplicity** - Does the component implementation assume a single service or does it explicitly handle multiple occurrences? For example, when a component uses the Log Service, it only needs to bind to one Log Service to function correctly. Alternatively, when the Configuration Admin uses the Configuration Listener services it needs to bind to all target services present in the service registry to dispatch its events correctly.

- **Optionality** - Can the component function without any bound service present? Some components can still perform useful tasks even when no target service is available, other components must bind to at least one target service before they can be useful. For example, the Configuration Admin in the previous example must still provide its functionality even if there are no Configuration Listener services present. Alternatively, an application that solely presents a Servlet page has little to do when the Http Service is not present, it should therefore use a reference with a mandatory cardinality.

The cardinality is expressed with the following syntax:

```
cardinality ::= optionality '..' multiplicity
optionality ::= '0' | '1'
multiplicity ::= '1' | 'n'
```

A reference is satisfied if the number of target services is equal to or more than the optionality. The multiplicity is irrelevant for the satisfaction of the reference. The multiplicity only specifies if the component implementation is written to handle being bound to multiple services (n) or requires SCR to select and bind to a single service (1).

The cardinality for a reference can be specified as one of four choices:

- **0..1** - Optional and unary.
- **1..1** - Mandatory and unary (Default).
- **0..n** - Optional and multiple.
- **1..n** - Mandatory and multiple.

When a satisfied component configuration is activated, there must be at most one bound service for each reference with a unary cardinality and at least one bound service for each reference with a mandatory cardinality. If the cardinality constraints cannot be maintained after a component configuration is activated, that is the reference becomes unsatisfied, the component configuration must be deactivated. If the reference has a unary cardinality and there is more than one target service for the reference, then the bound service must be the target service with the highest service ranking as specified by the service.ranking property. If there are multiple target services with the same service ranking, then the bound service must be the target service with the highest service ranking and the lowest service id as specified by the service.id property.

For example, a component wants to register a resource with all Http Services that are available. Such a scenario has the cardinality of **0..n**. The code must be prepared to handle multiple calls to the bind method for each Http Service in such a case. In this example, the code uses the registerResources method to register a directory for external access.

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component name="example.listen"
xmlns:scr="http://www.osgi.org/xmlns/scr/v1.2.0">
  <implementation class="com.acme.HttpResourceImpl"/>
  <reference name="HTTP"
    interface="org.osgi.service.http.HttpService"
    cardinality="0..n"/>
```
112.3.4 Reference Policy

Once all the references of a component are satisfied, a component configuration can be activated and therefore bound to target services. However, the dynamic nature of the OSGi service registry makes it likely that services are registered, modified and unregistered after target services are bound. These changes in the service registry could make one or more bound services no longer a target service thereby making obsolete any object references that the component has to these service objects. Components therefore must specify a policy how to handle these changes in the set of bound services. A policy-option can further refine how changes affect bound services.

The static policy is the most simple policy and is the default policy. A component instance never sees any of the dynamics. Component configurations are deactivated before any bound service for a reference having a static policy becomes unavailable. If a target service is available to replace the bound service which became unavailable, the component configuration must be reactivated and bound to the replacement service.

If the policy-option is reluctant then the registration of an additional target service for a reference must not result in deactivating and reactivating a component configuration. If the policy-option is greedy then the component must be reactivated when new applicable services become available, see Table 112.1 on page 178. A reference with a static policy is called a static reference. A static reference can still be updated dynamically if it specifies an updated method.

The static policy can be very expensive if it depends on services that frequently unregister and re-register or if the cost of activating and deactivating a component configuration is high. Static policy is usually also not applicable if the cardinality specifies multiple bound services.

The dynamic policy is slightly more complex since the component implementation must properly handle changes in the set of bound services that can occur on any thread at any time after the component instance is created. With the dynamic policy, SCR can change the set of bound services without deactivating a component configuration. If the component uses the event strategy to access services, then the component instance will be notified of changes in the set of bound services by calls to the bind, and unbind methods.

If the policy-option is reluctant then a bound reference is not rebound even if a more suitable service becomes available for a 1..1 or 0..1 reference. If the policy-option is greedy then the component must be unbound and rebound for that reference.

A reference with a dynamic policy is called a dynamic reference.

The previous example with the registering of a resource directory used a static policy. This implied that the component configurations are deactivated when there is a change in the bound set of Http Services. The code in the example can be seen to easily handle the dynamics of Http Services that come and go. The component description can therefore be updated to:

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
The code is identical to the previous example.

### 112.3.5 Policy Option

The policy-option defines how eager the reference is to rebind when a new, potentially with a higher ranking, target service becomes available. It can have the following values:

- **reluctant** - Minimize rebinding and reactivating.
- **greedy** - Maximize the use of the best service by deactivating static references or rebinding dynamic references.

Table 112.1 defines the actions that are taken when a better target service becomes available. In this context, better is when the reference is not bound or when the new target service has a higher ranking than the bound service.

<table>
<thead>
<tr>
<th>Cardinality</th>
<th>static reluctant</th>
<th>static greedy</th>
<th>dynamic reluctant</th>
<th>dynamic greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>Ignore</td>
<td>Reactivate to bind the better target service.</td>
<td>If no service is bound, bind to new target service. Otherwise, ignore new target service.</td>
<td>If no service is bound, bind to better target service. Otherwise, unbind the bound service and bind the better target service.</td>
</tr>
<tr>
<td>1..1</td>
<td>Ignore</td>
<td>Reactivate to bind the better target service.</td>
<td>Ignore</td>
<td>Unbind the bound service, then bind the new service.</td>
</tr>
<tr>
<td>0..n</td>
<td>Ignore</td>
<td>Reactivate</td>
<td>Bind new target service</td>
<td>Bind new target service</td>
</tr>
<tr>
<td>1..n</td>
<td>Ignore</td>
<td>Reactivate</td>
<td>Bind new target service</td>
<td>Bind new target service</td>
</tr>
</tbody>
</table>

### 112.3.6 Selecting Target Services

The target services for a reference are constrained by the reference's interface name and target property. By specifying a filter in the target property, the programmer and deployer can constrain the set of services that should be part of the target services.

For example, a component wants to track all Component Factory services that have a factory identification of `acme.application`. The following component description shows how this can be done.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scr:component name="example.listen"
    xmlns:scr="http://www.osgi.org/xmlns/scr/v1.2.0">
    <implementation class="com.acme.FactoryTracker"/>
</scr:component>
```
The filter is manifested as a component property called the target property. The target property can also be set by property and properties elements, see Property Element on page 182. The deployer can also set the target property by establishing a configuration for the component which sets the value of the target property. This allows the deployer to override the target property in the component description. See Component Properties on page 196 for more information.

112.3.7 Circular References

It is possible for a set of component descriptions to create a circular dependency. For example, if component A references a service provided by component B and component B references a service provided by component A then a component configuration of one component cannot be satisfied without accessing a partially activated component instance of the other component. SCR must ensure that a component instance is never accessible to another component instance or as a service until it has been fully activated, that is it has returned from its activate method if it has one.

Circular references must be detected by SCR when it attempts to satisfy component configurations and SCR must fail to satisfy the references involved in the cycle and log an error message with the Log Service, if present. However, if one of the references in the cycle has optional cardinality SCR must break the cycle. The reference with the optional cardinality can be satisfied and bound to zero target services. Therefore the cycle is broken and the other references may be satisfied.

112.4 Component Description

Component descriptions are defined in XML documents contained in a bundle and any attached fragments.

If SCR detects an error when processing a component description, it must log an error message with the Log Service, if present, and ignore the component description. Errors can include XML parsing errors and ill-formed component descriptions.

112.4.1 Annotations

A number of CLASS retention annotations have been provided to allow tools to construct the XML from the Java class files. These annotations will be discussed with the appropriate elements and attributes. Since the naming rules between XML and Java differ, some name changes are necessary.

- **Elements** - The annotation class that corresponds to an element starts with an upper case. For example the component element is represented by the @Component annotation.

- **Attributes** - Multi word attributes that use a minus sign ('-') are changed to camel case. For example, the component element configuration-pid attribute is the configurationPid member in the @Component annotation.

Some elements do not have a corresponding annotation since the annotations can be parameterized by the type information in the Java class. For example, the @Component annotation synthesizes the implement element's class attribute from the type it is applied to.

These annotations are intended to be used during build time to generate the XML and are not recognized by SCR at runtime.
112.4.2 Service Component Header

XML documents containing component descriptions must be specified by the Service-Component header in the manifest. The value of the header is a comma separated list of paths to XML entries within the bundle.

Service-Component ::= header // See Common Header Syntax in Core

The Service-Component header has no architected directives or properties. The header can be left empty.

The last component of each path in the Service-Component header may use wildcards so that Bundle.findEntries can be used to locate the XML document within the bundle and its fragments. For example:

Service-Component: OSGI-INF/*.xml

A Service-Component manifest header specified in a fragment is ignored by SCR. However, XML documents referenced by a bundle’s Service-Component manifest header may be contained in attached fragments.

SCR must process each XML document specified in this header. If an XML document specified by the header cannot be located in the bundle and its attached fragments, SCR must log an error message with the Log Service, if present, and continue.

112.4.3 XML Document

A component description must be in a well-formed XML document, Extensible Markup Language (XML) 1.0, stored in a UTF-8 encoded bundle entry. The namespace for component descriptions is:

http://www.osgi.org/xmlns/scr/v1.2.0

The recommended prefix for this namespace is scr. This prefix is used by examples in this specification. XML documents containing component descriptions may contain a single, root component element or one or more component elements embedded in a larger document. Use of the namespace for component descriptions is mandatory. The attributes and sub-elements of a component element are always unqualified.

If an XML document contains a single, root component element which does not specify a namespace, then the http://www.osgi.org/xmlns/scr/v1.0.0 namespace is assumed. Component descriptions using the http://www.osgi.org/xmlns/scr/v1.0.0 namespace must be treated according to version 1.0 of this specification.

SCR must parse all component elements in the namespace. Elements not in this namespace must be ignored. Ignoring elements that are not recognized allows component descriptions to be embedded in any XML document. For example, an entry can provide additional information about components. These additional elements are parsed by another sub-system.

See Component Description Schema on page 202 for component description schema.

112.4.4 Component Element

The component element specifies the component description. The following text defines the structure of the XML grammar using a form that is similar to the normal grammar used in OSGi specifications. In this case the grammar should be mapped to XML elements:

```
<component> ::= <implementation>
    ( <properties> | <property> )* 
    <service>
    <reference> * 
```
SCR must not require component descriptions to specify the elements in the order listed above and as required by the XML schema. SCR must allow other orderings since arbitrary orderings of these elements do not affect the meaning of the component description. Only the relative ordering of property and properties element have meaning.

The component element has the attributes and @Component annotations defined in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
<td>The name of a component must be unique within a bundle. The component name is used as a PID to retrieve component properties from the OSGi Configuration Admin service if present, unless a configuration-pid attribute has been defined. See Deployment on page 197 for more information. If the component name is used as a PID then it should be unique within the framework. The XML schema allows the use of component names which are not valid PIDs. Care must be taken to use a valid PID for a component name if the component should be configured by the Configuration Admin service. This attribute is optional. The default value of this attribute is the value of the class attribute of the nested implementation element. If multiple component elements in a bundle use the same value for the class attribute of their nested implementation element, then using the default value for this attribute will result in duplicate component names. In this case, this attribute must be specified with a unique value.</td>
</tr>
<tr>
<td>enabled</td>
<td>enabled</td>
<td>Controls whether the component is enabled when the bundle is started. The default value is true. If enabled is set to false, the component is disabled until the method enableComponent is called on the ComponentContext object. This allows some initialization to be performed by some other component in the bundle before this component can become satisfied. See Enabled on page 187.</td>
</tr>
<tr>
<td>factory</td>
<td>factory</td>
<td>If set to a non-empty string, it indicates that this component is a factory component. SCR must register a Component Factory service for each factory component. See Factory Component on page 171.</td>
</tr>
<tr>
<td>immediate</td>
<td>immediate</td>
<td>Controls whether component configurations must be immediately activated after becoming satisfied or whether activation should be delayed. The default value is false if the factory attribute or if the service element is specified and true otherwise. If this attribute is specified, its value must be false if the factory attribute is also specified or must be true unless the service element is also specified.</td>
</tr>
<tr>
<td>configuration-policy</td>
<td>configurationPolicy (OPTIONAL, REQUIRE, or IGNORE)</td>
<td>Controls whether component configurations must be satisfied depending on the presence of a corresponding Configuration object in the OSGi Configuration Admin service. A corresponding configuration is a Configuration object where the PID is the name of the component.</td>
</tr>
<tr>
<td>configuration-pid</td>
<td>configurationPid</td>
<td>The configuration PID to be used for the component in conjunction with Configuration Admin. The default value for this attribute is the name of the component, or if this is also not specified, the implementation class name.</td>
</tr>
</tbody>
</table>

The component element has the attributes and @Component annotations defined in the following table.
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activate</td>
<td>Activate</td>
<td>Specifies the name of the method to call when a component configuration is activated. The default value of this attribute is activate. See Activate Method on page 191 for more information. The annotation must be applied to the activate method and can only be used once.</td>
</tr>
<tr>
<td>deactivate</td>
<td>Deactivate</td>
<td>Specifies the name of the method to call when a component configuration is deactivated. The default value of this attribute is deactivate. See Deactivate Method on page 194 for more information. The annotation must be applied to the deactivate method and can only be used once.</td>
</tr>
<tr>
<td>modified</td>
<td>Modified</td>
<td>Specifies the name of the method to call when the configuration properties for a component configuration is using a Configuration object from the Configuration Admin service and that Configuration object is modified without causing the component configuration to become unsatisfied. If this attribute is not specified, then the component configuration will become unsatisfied if its configuration properties use a Configuration object that is modified in any way. See Modified Method on page 193 for more information. The annotation must be applied to the modified method and can only be used once.</td>
</tr>
</tbody>
</table>

### 112.4.5 Implementation Element

The implementation element is required and defines the name of the component implementation class. The single attribute is defined in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>Component</td>
<td>The Java fully qualified name of the implementation class. The component Component annotation will define the implementation element automatically from the type it is applied to.</td>
</tr>
</tbody>
</table>

The class is retrieved with the loadClass method of the component's bundle. The class must be public and have a public constructor without arguments (this is normally the default constructor) so component instances may be created by SCR with the newInstance method on Class.

If the component description specifies a service, the class must implement all interfaces that are provided by the service.

### 112.4.6 Property Element

A component description can define a number of properties. These can defined inline or from a resource in the bundle. The property and properties elements can occur multiple times and they can be interleaved. This interleaving is relevant because the properties are processed from top to bottom. Later properties override earlier properties that have the same name.

Properties can also be overridden by a Configuration Admin service's Configuration object before they are exposed to the component or used as service properties. This is described in Component Properties on page 196 and Deployment on page 197.

The property element has the attributes and annotations defined in the following table.
Table 112.4 Property Element and Annotations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Component property</td>
<td>The name of the property.</td>
</tr>
<tr>
<td>value</td>
<td></td>
<td>The value of the property. This value is parsed according to the property type. If the value attribute is specified, the body of the element is ignored. If the type of the property is not String, parsing of the value is done by the static valueOf(String) method in the given type. For Character types, the conversion must be handled by Integer.valueOf method, a Character is always represented by its Unicode value.</td>
</tr>
<tr>
<td>type</td>
<td></td>
<td>The type of the property. Defines how to interpret the value. The type must be one of the following Java types:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- String (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Long</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Float</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Character</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Short</td>
</tr>
</tbody>
</table>

The type attribute is not specified, the body of the property element must contain one or more values. The value of the property is then an array of the specified type. Except for String objects, the result will be translated to an array of primitive types. For example, if the type attribute specifies Integer, then the resulting array must be int[].

Values must be placed one per line and blank lines are ignored. Parsing of the value is done by the parse methods in the class identified by the type, after trimming the line of any beginning and ending white space. String values are also trimmed of beginning and ending white space before being placed in the array.

For example, a component that needs an array of hosts can use the following property definition:

```xml
<property name="hosts">
  www.acme.com
  backup.acme.com
</property>
```

This property declaration results in the property hosts, with a value of String[] { "www.acme.com", "backup.acme.com" }.

A property can also be set with the property annotation element of Component. This element is an array of strings that must follow the following syntax:

```java
@Component(property={"foo:Integer=1","foo:Integer=2","foo:Integer=3"})
```

In this case name, type, and value parts map to the attributes of the property element. If multiple values must be specified then the same name can be repeated multiple times. The annotation does not support ordering of properties. For example:
public class FooImpl {
  ...
}


The entry is read and processed to obtain the properties and their values. The properties element attributes are defined in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entry</td>
<td>Component properties</td>
<td>The entry path relative to the root of the bundle</td>
</tr>
</tbody>
</table>

For example, to include vendor identification properties that are stored in the OSGI-INF directory, the following definition could be used:

```xml
<properties entry="OSGI-INF/vendor.properties"/>
```

The Component properties element can be used to provide the same information, this element consists of an array of strings where each string defines an entry. The order within the array is the order that must be used for the XML. However, the annotations do not allow mixing of the property and properties element.

For example:

```java
@Component(properties="OSGI-INF/vendor.properties")
```

### 112.4.7 Service Element

The service element is optional. It describes the service information to be used when a component configuration is to be registered as a service.

A service element has the following attribute defined in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>servicefactory</td>
<td>Component servicefactory</td>
<td>Controls whether the service uses the ServiceFactory concept of the OSGi Framework. The default value is false. If servicefactory is set to true, a different component configuration is created, activated and its component instance returned as the service object for each distinct bundle that requests the service. Each of these component configurations has the same component properties. Otherwise, the same component instance from the single component configuration is returned as the service object for all bundles that request the service.</td>
</tr>
</tbody>
</table>

The servicefactory attribute must not be true if the component is a factory component or an immediate component. This is because SCR is not free to create component configurations as necessary to support servicefactory. A component description is ill-formed if it specifies that the component is a factory component or an immediate component and servicefactory is set to true.

The service element must have one or more provide elements that define the service interfaces. The provide element has the attribute defined in the following table.
Table 112.7  Provide Element and Annotations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
</table>
| interface | Component service | The name of the interface that this service is registered under. This name must be the fully qualified name of a Java class. For example, org.osgi.service.log.LogService. The specified Java class should be an interface rather than a class, however specifying a class is supported. The component implementation class must implement all the specified service interfaces. The Component annotation can specify the provided services, if this element is not specified all directly implemented interfaces on the component’s type are defined as service interfaces. Specifying an empty array indicates that no service should be registered. For example, a component implements an Event Handler service.  

```xml
<service>
  <provide interface="org.osgi.service.eventadmin.EventHandler"/>
</service>
```

This previous example can be generated with the following annotation:

```java
@Component
public class Foo implements EventualHandler { ... }
```

### 112.4.8 Reference Element

A reference declares a dependency that a component has on a set of target services. A component configuration is not satisfied, unless all its references are satisfied. A reference specifies target services by specifying their interface and an optional target filter.

A reference element has the attributes defined in the following table.

Table 112.8  Reference Element and Annotations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
<td>The name of the reference. This name is local to the component and can be used to locate a bound service of this reference with one of the locateService methods of ComponentContext. Each reference element within the component must have a unique name. This name attribute is optional. The default value of this attribute is the value of the interface attribute of this element. If multiple reference elements in the component use the same interface name, then using the default value for this attribute will result in duplicate reference names. In this case, this attribute must be specified with a unique name for the reference to avoid an error. The Reference annotation will use the name of the annotated method as the default reference name. If the method name begins with bind, set or add, that prefix is removed. Fully qualified name of the class that is used by the component to access the service. The service provided to the component must be type compatible with this class. That is, the component must be able to cast the service object to this class. A service must be registered under this name to be considered for the set of target services. The Reference annotation will use the type of the first argument of the method it is applied for the service value.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Annotation</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cardinality</td>
<td>cardinality</td>
<td>Specifies if the reference is optional and if the component implementation support a single bound service or multiple bound services. See Reference Cardinality on page 176.</td>
</tr>
<tr>
<td>policy</td>
<td>policy</td>
<td>The policy declares the assumption of the component about dynamicity. See Reference Policy on page 177.</td>
</tr>
<tr>
<td>policy-option</td>
<td>policyOption</td>
<td>Defines the policy when a better service becomes available. See Reference Policy on page 177.</td>
</tr>
<tr>
<td>target</td>
<td>target</td>
<td>An optional OSGi Framework filter expression that further constrains the set of target services. The default is no filter, limiting the set of matched services to all service registered under the given reference interface. The value of this attribute is used to set a target property. See Selecting Target Services on page 178.</td>
</tr>
<tr>
<td>bind</td>
<td>Reference</td>
<td>The name of a method in the component implementation class that is used to notify that a service is bound to the component configuration. For static references, this method is only called before the activate method. For dynamic references, this method can also be called while the component configuration is active. See Accessing Services on page 173. The Reference annotation will use the method it is applied to as the bind method.</td>
</tr>
<tr>
<td>updated</td>
<td>updated</td>
<td>The name of a method in the component implementation class that is used to notify that a bound service has modified its properties.</td>
</tr>
<tr>
<td>unbind</td>
<td>unbind</td>
<td>Same as bind, but is used to notify the component configuration that the service is unbound. For static references, the method is only called after the deactivate method. For dynamic references, this method can also be called while the component configuration is active. See Accessing Services on page 173.</td>
</tr>
</tbody>
</table>

The following code demonstrates the use of the Reference annotation.

```java
@Component
public class FooImpl implements Foo {
    @Activate
    void open() { ... }
    @Deactivate
    void close() { ... }
    @Reference(
        policy = DYNAMIC,
        policyOption=GREEDY,
        cardinality=MANDATORY
    )
    void setLog( LogService log) { ... }
    void unsetLog( LogService log) { ... }
    void updatedLog( Map<String,?> ref ) { ... }
}
```
112.5 Component Life Cycle

112.5.1 Enabled

A component must first be enabled before it can be used. A component cannot be enabled unless the component's bundle is started. See Starting Bundles in OSGi Core Release 6. All components in a bundle become disabled when the bundle is stopped. So the life cycle of a component is contained within the life cycle of its bundle.

Every component can be enabled or disabled. The initial enabled state of a component is specified in the component description via the enabled attribute of the component element. See Component Element on page 180. Component configurations can be created, satisfied and activated only when the component is enabled.

The enabled state of a component can be controlled with the Component Context enableComponent(String) and disableComponent(String) methods. The purpose of later enabling a component is to be able to decide programmatically when a component can become enabled. For example, an immediate component can perform some initialization work before other components in the bundle are enabled. The component descriptions of all other components in the bundle can be disabled by having enabled set to false in their component descriptions. After any necessary initialization work is complete, the immediate component can call enableComponent to enable the remaining components.

The enableComponent and disableComponent methods must return after changing the enabled state of the named component. Any actions that result from this, such as activating or deactivating a component configuration, must occur asynchronously to the method call. Therefore a component can disable itself.

All components in a bundle can be enabled by passing a null as the argument to enableComponent.

112.5.2 Satisfied

Component configurations can only be activated when the component configuration is satisfied. A component configuration becomes satisfied when the following conditions are all satisfied:

- The component is enabled.
- If the component description specifies configuration-policy=required, then a Configuration object for the component is present in the Configuration Admin service.
- Using the component properties of the component configuration, all the component's references are satisfied. A reference is satisfied when the reference specifies optional cardinality or there is at least one target service for the reference.

Once any of the listed conditions are no longer true, the component configuration becomes unsatisfied. An activated component configuration that becomes unsatisfied must be deactivated.

112.5.3 Immediate Component

A component is an immediate component when it must be activated as soon as its dependencies are satisfied. Once the component configuration becomes unsatisfied, the component configuration must be deactivated. If an immediate component configuration is satisfied and specifies a service, SCR must register the component configuration as a service in the service registry and then activate the component configuration. The service properties for this registration consist of the component properties as defined in Service Properties on page 197.

The state diagram is shown in Figure 112.2.
112.5.4 Delayed Component

A key attribute of a delayed component is the delaying of class loading and object creation. Therefore, the activation of a delayed component configuration does not occur until there is an actual request for a service object. A component is a delayed component when it specifies a service but it is not a factory component and does not have the immediate attribute of the component element set to true.

SCR must register a service after the component configuration becomes satisfied. The registration of this service must look to observers of the service registry as if the component's bundle actually registered this service. This strategy makes it possible to register services without creating a class loader for the bundle and loading classes, thereby allowing reduction in initialization time and a delay in memory footprint.

When SCR registers the service on behalf of a component configuration, it must avoid causing a class load to occur from the component's bundle. SCR can ensure this by registering a ServiceFactory object with the Framework for that service. By registering a ServiceFactory object, the actual service object is not needed until the ServiceFactory is called to provide the service object. The service properties for this registration consist of the component properties as defined in Service Properties on page 197.

The activation of a component configuration must be delayed until its service is requested. When the service is requested, if the service has the servicefactory attribute set to true, SCR must create and activate a unique component configuration for each bundle requesting the service. Otherwise, SCR must activate a single component configuration which is used by all bundles requesting the service. A component instance can determine the bundle it was activated for by calling the getUsingBundle() method on the Component Context.

The activation of delayed components is depicted in a state diagram in Figure 112.3. Notice that multiple component configurations can be created from the REGISTERED state if a delayed component specifies servicefactory set to true.

If the service registered by a component configuration becomes unused because there are no more bundles using it, then SCR should deactivate that component configuration. This allows SCR implementations to eagerly reclaim activated component configurations.
### Factory Component

SCR must register a Component Factory service as soon as the `component factory` becomes satisfied. The component factory is satisfied when the following conditions are all satisfied:

- The component is enabled.
- Using the component properties specified by the component description, all the component’s references are satisfied. A reference is satisfied when the reference specifies optional cardinality or there is at least one target service for the reference.

The component factory, however, does not use any of the target services and does not bind to them.

Once any of the listed conditions are no longer true, the component factory becomes unsatisfied and the Component Factory service must be unregistered. Any component configurations activated via the component factory are unaffected by the unregistration of the Component Factory service, but may themselves become unsatisfied for the same reason.

The Component Factory service must be registered under the name `org.osgi.service.component.ComponentFactory` with the following service properties:

- `component.name` - The name of the component.
- `component.factory` - The value of the factory attribute.

The service properties of the Component Factory service must not include the component properties.

New component configurations are created and activated when the `newInstance` method of the Component Factory service is called. If the component description specifies a service, the component configuration is registered as a service under the provided interfaces. The service properties for this registration consist of the component properties as defined in `Service Properties` on page 197. The service registration must take place before the component configuration is activated. Service unregistration must take place before the component configuration is deactivated.
A Component Factory service has a single method: `newInstance(Dictionary)`. This method must create, satisfy and activate a new component configuration and register its component instance as a service if the component description specifies a service. It must then return a `ComponentInstance` object. This `ComponentInstance` object can be used to get the component instance with the `getInstance()` method.

SCR must attempt to satisfy the component configuration created by `newInstance` before activating it. If SCR is unable to satisfy the component configuration given the component properties and the Dictionary argument to `newInstance`, the `newInstance` method must throw a `ComponentException`.

The client of the Component Factory service can also deactivate a component configuration with the `dispose()` method on the `ComponentInstance` object. If the component configuration is already deactivated, or is being deactivated, then this method is ignored. Also, if the component configuration becomes unsatisfied for any reason, it must be deactivated by SCR.

Once a component configuration created by the Component Factory has been deactivated, that component configuration will not be reactivated or used again.

**112.5.6 Activation**

Activating a component configuration consists of the following steps:

1. Load the component implementation class.
2. Create the component instance and component context.
3. Bind the target services. See `Binding Services` on page 191.
4. Call the activate method, if present. See `Activate Method` on page 191.

Component instances must never be reused. Each time a component configuration is activated, SCR must create a new component instance to use with the activated component configuration. A component instance must complete activation before it can be deactivated. Once the component configuration is deactivated or fails to activate due to an exception, SCR must unbind all the component’s bound services and discard all references to the component instance associated with the activation.
112.5.7 Binding Services

When a component configuration's reference is satisfied, there is a set of zero or more target services for that reference. When the component configuration is activated, a subset of the target services for each reference are bound to the component configuration. The subset is chosen by the cardinality of the reference. See Reference Cardinality on page 176.

When binding services, the references are processed in the order in which they are specified in the component description. That is, target services from the first specified reference are bound before services from the next specified reference.

For each reference using the event strategy, the bind method must be called for each bound service of that reference. This may result in activating a component configuration of the bound service which could result in an exception. If the loss of the bound service due to the exception causes the reference's cardinality constraint to be violated, then activation of this component configuration will fail. Otherwise the bound service which failed to activate will be considered unbound. If a bind method throws an exception, SCR must log an error message containing the exception with the Log Service, if present, but the activation of the component configuration does not fail.

112.5.8 Activate Method

A component instance can have an activate method. The name of the activate method can be specified by the activate attribute. See Component Element on page 180. If the activate attribute is not specified, the default method name of activate is used. The prototype of the activate method is:

```java
void <method-name>(<arguments>);
```

The activate method can take zero or more arguments. Each argument must be of one of the following types:

- `ComponentContext` - The component instance will be passed the Component Context for the component configuration.
- `BundleContext` - The component instance will be passed the Bundle Context of the component's bundle.
- `Map` - The component instance will be passed an unmodifiable Map containing the component properties.

A suitable method is selected using the following priority:

1. The method takes a single argument and the type of the argument is `org.osgi.service.component.ComponentContext`.
2. The method takes a single argument and the type of the argument is `org.osgi.framework.BundleContext`.
3. The method takes a single argument and the type of the argument is the `java.util.Map`.
4. The method takes two or more arguments and the type of each argument must be `org.osgi.service.component.ComponentContext`, `org.osgi.framework.BundleContext` or `java.util.Map`. If multiple methods match this rule, this implies the method name is overloaded and SCR may choose any of the methods to call.
5. The method takes zero arguments.

When searching for the activate method to call, SCR must locate a suitable method as specified in Locating Component Methods on page 201. If the activate attribute is specified and no suitable method is located, SCR must log an error message with the Log Service, if present, and the component configuration is not activated.

If an activate method is located, SCR must call this method to complete the activation of the component configuration. If the activate method throws an exception, SCR must log an error message con-
112.5.9 **Component Context**

The Component Context is made available to a component instance via the `activate` and `deactivate` methods. It provides the interface to the execution context of the component, much like the Bundle Context provides a bundle the interface to the Framework. A Component Context should therefore be regarded as a capability and not shared with other components or bundles.

Each distinct component instance receives a unique Component Context. Component Contexts are not reused and must be discarded when the component configuration is deactivated.

112.5.10 **Bound Service Replacement**

If an active component configuration has a dynamic reference with unary cardinality and the bound service is modified or unregistered and ceases to be a target service, or the policy-option is `greedy` and a better target service becomes available then SCR must attempt to replace the bound service with a new target service. SCR must first bind a replacement target service and then unbind the outgoing service. This reversed order allows the component to not having to handle the inevitable gap between the unbind and bind methods. However, this means that in the unbind method care must be taken to not overwrite the newly bound service. For example, the following code handles the associated concurrency issues and simplify handling the reverse order.

```java
final AtomicReference<LogService> log = new AtomicReference<LogService>();

void setLogService( LogService log ) {
    this.log.set(log);
}

void unsetLogService( LogService log ) {
    this.log.compareAndSet(log, null);
}
```

If the dynamic reference has a mandatory cardinality and no replacement target service is available, the component configuration must be deactivated because the cardinality constraints will be violated.

If a component configuration has a static reference and a bound service is modified or unregistered and ceases to be a target service, or the policy-option is `greedy` and a better target service becomes available then SCR must deactivate the component configuration. Afterwards, SCR must attempt to activate the component configuration again if another target service can be used as a replacement for the outgoing service.

112.5.11 **Updated method**

If an active component is bound to a service that modifies it properties then the component can be notified with the update method specified on the reference element. This method can be called with a Service Reference or a Map to supply the updated service properties.

112.5.12 **Modification**

Modifying a component configuration can occur if the component description specifies the modified attribute and the component properties of the component configuration use a Configuration object from the Configuration Admin service and that Configuration object is modified without causing the component configuration to become unsatisfied. If this occurs, the component instance will be notified of the change in the component properties.
If the modified attribute is not specified, then the component configuration will become unsatisfied if its component properties use a Configuration object and that Configuration object is modified in any way.

Modifying a component configuration consists of the following steps:

1. Update the component context for the component configuration with the modified configuration properties.
2. Call the modified method. See Modified Method on page 193.
3. Modify the bound services for the dynamic references if the set of target services changed due to changes in the target properties. See Bound Service Replacement on page 192.
4. If the component configuration is registered as a service, modify the service properties.

A component instance must complete activation, or a previous modification, before it can be modified.

See Configuration Changes on page 198 for more information.

112.5.13 Modified Method

The name of the modified method is specified by the modified attribute. See Component Element on page 180. The prototype and selection priority of the modified method is identical to that of the activate method. See Activate Method on page 191.

SCR must locate a suitable method as specified in Locating Component Methods on page 201. If the modified attribute is specified and no suitable method is located, SCR must log an error message with the Log Service, if present, and the component configuration becomes unsatisfied and is deactivated as if the modified attribute was not specified.

If a modified method is located, SCR must call this method to notify the component configuration of changes to the component properties. If the modified method throws an exception, SCR must log an error message containing the exception with the Log Service, if present and continue processing the modification.

112.5.14 Deactivation

Deactivating a component configuration consists of the following steps:

1. Call the deactivate method, if present. See Deactivate Method on page 194.
2. Unbind any bound services. See Unbinding on page 194.
3. Release all references to the component instance and component context.

A component instance must complete activation or modification before it can be deactivated. A component configuration can be deactivated for a variety of reasons. The deactivation reason can be received by the deactivate method. The following reason values are defined:

- 0 - Unspecified.
- 1 - The component was disabled.
- 2 - A reference became unsatisfied.
- 3 - A configuration was changed.
- 4 - A configuration was deleted.
- 5 - The component was disposed.
- 6 - The bundle was stopped.

Once the component configuration is deactivated, SCR must discard all references to the component instance and component context associated with the activation.
### 112.5.15 Deactivate Method

A component instance can have a deactivate method. The name of the deactivate method can be specified by the `deactivate` attribute. See Component Element on page 180. If the `deactivate` attribute is not specified, the default method name of `deactivate` is used. The prototype of the deactivate method is:

```java
void <method-name>(<arguments>);
```

The deactivate method can take zero or more arguments. Each argument must be assignable from one of the following types:

- `ComponentContext` - The component instance will be passed the Component Context for the component.
- `BundleContext` - The component instance will be passed the Bundle Context of the component's bundle.
- `Map` - The component instance will be passed an unmodifiable Map containing the component properties.
- `int` or `Integer` - The component instance will be passed the reason the component configuration is being deactivated. See Deactivation on page 193.

A suitable method is selected using the following priority:

1. The method takes a single argument and the type of the argument is `org.osgi.service.component.ComponentContext`.
2. The method takes a single argument and the type of the argument is `org.osgi.framework.BundleContext`.
3. The method takes a single argument and the type of the argument is the `java.util.Map`.
4. The method takes a single argument and the type of the argument is the `int`.
5. The method takes a single argument and the type of the argument is the `java.lang.Integer`.
6. The method takes two or more arguments and the type of each argument must be `org.osgi.service.component.ComponentContext`, `org.osgi.framework.BundleContext`, `java.util.Map`, `int` or `java.lang.Integer`. If multiple methods match this rule, this implies the method name is overloaded and SCR may choose any of the methods to call.
7. The method takes zero arguments.

When searching for the deactivate method to call, SCR must locate a suitable method as specified in Locating Component Methods on page 201. If the `deactivate` attribute is specified and no suitable method is located, SCR must log an error message with the Log Service, if present, and the deactivation of the component configuration will continue.

If a deactivate method is located, SCR must call this method to commence the deactivation of the component configuration. If the deactivate method throws an exception, SCR must log an error message containing the exception with the Log Service, if present, and the deactivation of the component configuration will continue.

### 112.5.16 Unbinding

When a component configuration is deactivated, the bound services are unbound from the component configuration.

When unbinding services, the references are processed in the reverse order in which they are specified in the component description. That is, target services from the last specified reference are unbound before services from the previous specified reference.

For each reference using the event strategy, the unbind method must be called for each bound service of that reference. If an unbind method throws an exception, SCR must log an error message...
containing the exception with the Log Service, if present, and the deactivation of the component configuration will continue.

### 112.5.17 Life Cycle Example

A component could declare a dependency on the Http Service to register some resources.

```xml
<scr:component name="example.binding"
    xmlns:scr="http://www.osgi.org/xmlns/scr/v1.2.0">
    <implementation class="example.Binding"/>
    <reference name="LOG"
        interface="org.osgi.service.log.LogService"
        cardinality="1..1"
        policy="static"/>
    <reference name="HTTP"
        interface="org.osgi.service.http.HttpService"
        cardinality="0..1"
        policy="dynamic"
        bind="setHttp"
        unbind="unsetHttp"/>
</scr:component>
```

The component implementation code looks like:

```java
public class Binding {
    LogService log;
    HttpService http;

    private void setHttp(HttpService h) {
        this.http = h;
        // register servlet
    }
    private void unsetHttp(HttpService h) {
        this.h = null;
        // unregister servlet
    }
    private void activate(ComponentContext context) {
        log = (LogService) context.locateService("LOG");
    }
    private void deactivate(ComponentContext context) {
    }
}
```

This example is depicted in a sequence diagram in Figure 112.5. with the following scenario:

1. A bundle with the `example.Binding` component is started. At that time there is a Log Service h₁ and a Http Service h₁ registered.
2. The Http Service h₁ is unregistered
3. A new Http Service h₂ is registered
4. The Log Service h₁ is unregistered.
Each component configuration is associated with a set of component properties. The component properties are specified in the following places (in order of precedence):

1. Properties specified in the argument of ComponentFactory.newInstance method. This is only applicable for factory components.

2. Properties retrieved from the OSGi Configuration Admin service with a Configuration object that has a PID equal to the configuration PID. The configuration PID is the component name, or when specified, the configuration-pid attribute.

3. Properties specified in the component description. Properties specified later in the component description override properties that have the same name specified earlier. Properties can be specified in the component description in the following ways:
   - target attribute of reference elements - Sets a component property called the target property of the reference. The key of a target property is the name of the reference appended with .target. The value of a target property is the value of the target attribute. For example, a reference with the name http whose target attribute has the value "(http.port=80)" results in the component property having the name http.target and value "(http.port=80)". See Selecting Target Services on page 178. The target property can also be set wherever component properties can be set.
   - property and properties elements - See Property Element on page 182.

The precedence behavior allows certain default values to be specified in the component description while allowing properties to be replaced and extended by:

- A configuration in Configuration Admin
- The argument to ComponentFactory.newInstance method

SCR always adds the following component properties, which cannot be overridden:

- component.name - The component name.
- **component.id** - A unique value (Long) that is larger than all previously assigned values. These values are not persistent across restarts of SCR.

### 112.6.1 Service Properties

When SCR registers a service on behalf of a component configuration, SCR must follow the recommendations in *Property Propagation* on page 71 and must not propagate private configuration properties. That is, the service properties of the registered service must be all the component properties of the component configuration whose property names do not start with full stop (`.\u002E`).

Component properties whose names start with full stop are available to the component instance but are not available as service properties of the registered service.

### 112.7 Deployment

A component description contains default information to select target services for each reference. However, when a component is deployed, it is often necessary to influence the target service selection in a way that suits the needs of the deployer. Therefore, SCR uses Configuration objects from Configuration Admin to replace and extend the component properties for a component configuration. That is, through Configuration Admin, a deployer can configure component properties.

The configuration PID is used as the key for obtaining additional component properties from Configuration Admin. The following situations can arise:

- **No Configuration** - If the component's configuration-policy is set to ignore or there is no Configuration with a PID or factory PID equal to the configuration PID, then component configurations will not obtain component properties from Configuration Admin. Only component properties specified in the component description or via the `ComponentFactory.newInstance` method will be used.

- **Not Satisfied** - If the component's configuration-policy is set to require and there is no Configuration with a PID or factory PID equal to the configuration PID, then the component configuration is not satisfied and will not be activated.

- **Single Configuration** - If there exists a Configuration with a PID equal to the configuration PID, then component configurations will obtain additional component properties from Configuration Admin.

- **Factory Configuration** - If a factory PID exists, with zero or more Configurations, that is equal to the configuration PID, then for each Configuration, a component configuration must be created that will obtain additional component properties from Configuration Admin.

A factory configuration must not be used if the component is a factory component. This is because SCR is not free to create component configurations as necessary to support multiple Configurations. When SCR detects this condition, it must log an error message with the Log Service, if present, and ignore the component description.

SCR must obtain the Configuration objects from the Configuration Admin service using the Bundle Context of the bundle containing the component.

For example, there is a component named `com.acme.client` with a reference named `HTTP` that requires an Http Service which must be bound to a component `com.acme.httpserver` which provides an Http Service. A deployer can establish the following configuration:

```java
[PID=com.acme.client, factoryPID=null]
HTTP.target = (component.name=com.acme.httpserver)
```
112.7.1 Configuration Changes

SCR must track changes in the Configuration objects matching the configuration PID of the component description. Changes include the creating, updating and deleting of matching Configuration objects. The actions SCR must take when a configuration change for a component description occurs are based upon how the configuration-policy and modified attributes are specified in the component description, whether a component configuration becomes satisfied, remains satisfied or becomes unsatisfied and the type and number of matching Configuration objects.

112.7.1.1 Ignore Configuration Policy

For configuration-policy of ignore, component configurations are unaffected by configuration changes since the component properties do not include properties from Configuration objects.

112.7.1.2 Require Configuration Policy

For configuration-policy of require, component configurations require a Configuration object. With a factory configuration, there can be zero or more matching Configuration objects which will result in a component configuration for each Configuration object. With a factory component, multiple component configurations can be created all using a single matching Configuration object.

Deleting a Configuration object will cause the component configurations using it to become unsatisfied. Updating a Configuration object can cause the component configurations using it to become unsatisfied if any of the following occur:

- A target property change results in a bound service of a static reference ceasing to be a target service.
- A target property change results in unbound target services for a static reference with the greedy policy option.
- A target property change results in there being zero target services for a mandatory dynamic reference.
- The component description does not specify the modified attribute.

If a component configuration becomes unsatisfied, SCR must deactivate the component configuration. If the component configuration was not created from a factory component, SCR must attempt to satisfy the component configuration with the updated component properties.

If a component configuration remains satisfied and had been activated, the modified method is called. See Modification on page 192 for more information. If a component configuration remains satisfied but has not been activated, then, if the component configuration is registered as a service, SCR must modify the service properties.

112.7.1.3 Optional Configuration Policy

For configuration-policy of optional, component configurations do not require a Configuration object. Since a matching Configuration object is optional, component configurations can be satisfied when there is no matching Configuration object. If a matching Configuration object is then created, this is a configuration change for the component configurations that are not using a Configuration object. When there is only a single matching Configuration object which is then deleted, this is a configuration change for the component configurations using the deleted Configuration object.

With a factory configuration, there can be zero or more matching Configuration objects which will result in a component configuration for each Configuration object as well as a single component configuration when there are zero matching Configuration objects. With a factory component, multiple component configurations can be created all using either a single matching Configuration object or no Configuration object when there is no matching Configuration object.

A configuration change can cause a component configuration to become unsatisfied if any of the following occur:
• A target property change results in a bound service of a static reference ceasing to be a target service.
• A target property change results in unbound target services for a static reference with the greedy policy option.
• A target property change results in there being zero target services for a mandatory dynamic reference.
• The component description does not specify the modified attribute.

If a component configuration becomes unsatisfied, SCR must deactivate the component configuration. If the component configuration was not created from a factory component, SCR must attempt to satisfy the component configuration with the updated component properties.

If a component configuration remains satisfied and had been activated, the modified method is called. See Modification on page 192 for more information. If a component configuration remains satisfied but has not been activated, then, if the component configuration is registered as a service, SCR must modify the service properties.

### 112.8 Use of the Annotations

The Declarative Services Annotations provide a convenient way to create the component description XML during build time. Since annotations are placed in the source file and can use types, fields, and methods they can significantly simplify the use of Declarative Services.

The Declarative Services Annotations are build time annotations because one of the key aspect of Declarative Services is its laziness. Implementations can easily read the component description XML from the bundle, pre-process it, and cache the results between framework invocations. This is way it is unnecessary to create a class on the bundle when the bundle is started and/or scan the classes for annotations.

The Declarative Services Annotations are not inherited, they can only be used on a given class, annotations on its super class hierarchy or interfaces are not taken into account.

The primary annotation is the `Component` annotation. It indicates that a class is a component. It’s defaults create the easiest to use component:

• Its name is the class name
• It registers all directly implemented interfaces as services
• The instance will be shared by all bundles
• It is enabled
• It is immediate if it has no services, otherwise it is delayed
• It has an optional configuration policy
• The configuration PID is the class name

For example, the following class registers a Speech service that can run on a Macintosh:

```java
public interface Speech {
    void say(String what) throws Exception;
}

@Component
public class MacSpeech implements Speech {
    ScriptEngine engine =
        new ScriptEngineManager().getEngineByName("AppleScript");

    public void say(String message) throws Exception {
        // Use AppleScript to speak the message...
    }
}
```
The previous example must generate the following XML:

```xml
<scr:component name='com.example.MacSpeech'>
  <implementation class='com.example.MacSpeech'/>
  <service>
    <provide interface='com.example.service.speech.Speech'/>
  </service>
</component>
```

It is possible to add activate and deactivate methods on the component with the `Activate` and `Deactivate` annotations. If the component wants to be updated for changes in the configuration properties than it can also indicated the modified method with the `Modified` annotation. For example:

```java
@Activate
void open(Map<String,?> properties) { ... }

@Deactivate
void close() { ... }

@Modified
void modified(Map<String,?> properties) { ... }
```

If a component has dependencies on other services then they can be referenced with the `Reference` annotation that is applied to the bind method. The defaults for the reference annotations are:

- The name of the bind method is used for the name of the reference.
- 1:1 Cardinality.
- Static reluctant policy.
- The requested service is the type of the first argument of the method the `Reference` annotation is applied to.
- It will infer a default unset method and updated method based on the method name it is applied to.

For example:

```java
@Reference(cardinality=MULTIPLE, policy=DYNAMIC)
void setLogService( LogService log, Map<String,?> props) { ... }
void unsetLogService( LogService log ) { ... }
void updatedLogService( Map<String,?> map ) { ... }
```

## 112.9 Service Component Runtime

### 112.9.1 Relationship to OSGi Framework

The SCR must have access to the Bundle Context of any bundle that contains a component. The SCR needs access to the Bundle Context for the following reasons:

- To be able to register and get services on behalf of a bundle with components.
- To interact with the Configuration Admin on behalf of a bundle with components.
- To provide a component its Bundle Context when the Component Context `getBundleContext` method is called.
The SCR should use the `Bundle.getBundleContext()` method to obtain the Bundle Context reference.

### 112.9.2 Starting and Stopping SCR

When SCR is implemented as a bundle, any component configurations activated by SCR must be deactivated when the SCR bundle is stopped. When the SCR bundle is started, it must process any components that are declared in bundles that are started. This includes bundles which are started and are awaiting lazy activation.

### 112.9.3 Logging Error Messages

When SCR must log an error message to the Log Service, it must use a Log Service obtained using the component’s Bundle Context so that the resulting Log Entry is associated with the component’s bundle.

If SCR is unable to obtain, or use, a Log Service using the component’s Bundle Context, then SCR must log the error message to a Log Service obtained using SCR’s bundle context to ensure the error message is logged.

### 112.9.4 Locating Component Methods

SCR will need to locate activate, deactivate, modified, bind, updated, and unbind methods for a component instance. These methods will be located, and called, using reflection. The declared methods of each class in the component implementation class' hierarchy are examined for a suitable method. If a suitable method is found in a class, and it is accessible to the component implementation class, then that method must be used. If suitable methods are found in a class but none of the suitable methods are accessible by the component implementation class, then the search for suitable methods terminates with no suitable method having been located. If no suitable methods are found in a class, the search continues in the superclass.

Only methods that are accessible to the component implementation class will be used. If the method has the public or protected access modifier, then access is permitted. Otherwise, if the method has the private access modifier, then access is permitted only if the method is declared in the component implementation class. Otherwise, if the method has default access, also known as package private access, then access is permitted only if the method is declared in the component implementation class or if the method is declared in a superclass and all classes in the hierarchy from the component implementation class to the superclass, inclusive, are in the same package and loaded by the same class loader.

It is recommended that these methods should not be declared with the public access modifier so that they do not appear as public methods on the component instance when it is used as a service object. Having these methods declared public allows any code to call the methods with reflection, even if a Security Manager is installed. These methods are generally intended to only be called by SCR.

### 112.9.5 Bundle Activator Interaction

A bundle containing components may also declare a Bundle Activator. Such a bundle may also be marked for lazy activation. Since components are activated by SCR and Bundle Activators are called by the OSGi Framework, a bundle using both components and a Bundle Activator must take care. The Bundle Activator’s start method must not rely upon SCR having activated any of the bundle’s components. However, the components can rely upon the Bundle Activator’s start method having been called. That is, there is a happens-before relationship between the Bundle Activator’s start method being run and the components being activated.
112.10 Security

112.10.1 Service Permissions

Declarative services are built upon the existing OSGi service infrastructure. This means that Service Permission applies regarding the ability to publish, find or bind services.

If a component specifies a service, then component configurations for the component cannot be satisfied unless the component's bundle has ServicePermission[<provides>, REGISTER] for each provided interface specified for the service.

If a component's reference does not specify optional cardinality, the reference cannot be satisfied unless the component's bundle has ServicePermission[<interface>, GET] for the specified interface in the reference. If the reference specifies optional cardinality but the component's bundle does not have ServicePermission[<interface>, GET] for the specified interface in the reference, no service must be bound for this reference.

If a component is a factory component, then the above Service Permission checks still apply. But the component's bundle is not required to have ServicePermission[ComponentFactory, REGISTER] as the Component Factory service is registered by SCR.

112.10.2 Required Admin Permission

The SCR requires AdminPermission[*, CONTEXT] because it needs access to the bundle's Bundle Context object with the Bundle.getBundleContext() method.

112.10.3 Using hasPermission

SCR does all publishing, finding and binding of services on behalf of the component using the Bundle Context of the component's bundle. This means that normal stack-based permission checks will check SCR and not the component's bundle. Since SCR is registering and getting services on behalf of a component's bundle, SCR must call the Bundle.hasPermission method to validate that a component's bundle has the necessary permission to register or get a service.

112.11 Component Description Schema

This XML Schema defines the component description grammar.

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:scr="http://www.osgi.org/xmlns/scr/v1.2.0"
 targetNamespace="http://www.osgi.org/xmlns/scr/v1.2.0"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified"
 version="1.2.1">
  <annotation>
    <documentation xml:lang="en">
      This is the XML Schema for component descriptions used by the Service Component Runtime (SCR). Component description documents may be embedded in other XML documents. SCR will process all XML documents listed in the Service-Component manifest header of a bundle. XML documents containing component descriptions may contain a single, root component element or one or more component elements embedded in a larger document. Use of the namespace for component descriptions is mandatory. The attributes and subelements of a component element are always unqualified.
    </documentation>
  </annotation>
  <element name="component" type="scr:Tcomponent" />
  <complexType name="Tcomponent">
    <sequence>
      
    </sequence>
  </complexType>
</schema>
```
Implementations of SCR must not require component descriptions to specify the subelements of the component element in the order as required by the schema. SCR implementations must allow other orderings since arbitrary orderings do not affect the meaning of the component description. Only the relative ordering of property and properties element have meaning.

The default value of this attribute is the value of the class attribute of the nested implementation element. If multiple component elements use the same value for the class attribute of their nested implementation element, then using the default value for this attribute will result in duplicate names. In this case, this attribute must be specified with a unique value.
<complexType name="Tproperties">
<sequence>
  <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
</sequence>
<attribute name="entry" type="string" use="required" />
<anyAttribute processContents="lax" />
</complexType>

<complexType name="Tservice">
<sequence>
  <element name="provide" type="scr:Tprovide" minOccurs="1" maxOccurs="unbounded" />
  <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
</sequence>
<attribute name="servicefactory" type="boolean" default="false" use="optional" />
<anyAttribute processContents="lax" />
</complexType>

<complexType name="Tprovide">
<sequence>
  <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
</sequence>
<attribute name="interface" type="token" use="required" />
<anyAttribute processContents="lax" />
</complexType>

<complexType name="Treference">
<sequence>
  <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
</sequence>
<attribute name="name" type="token" use="optional">
  <annotation>
    <documentation xml:lang="en">
      The default value of this attribute is the value of the interface attribute of this element. If multiple instances of this element within a component element use the same value for the interface attribute, then using the default value for this attribute will result in duplicate names. In this case, this attribute must be specified with a unique value.
    </documentation>
  </annotation>
  <attribute name="interface" type="token" use="required" />
  <attribute name="cardinality" type="scr:Tcardinality" default="1..1" use="optional" />
  <attribute name="policy" type="scr:Tpolicy" default="static" use="optional" />
  <attribute name="policy-option" type="scr:Tpolicy-option" default="reluctant" use="optional" />
  <attribute name="target" type="string" use="optional" />
  <attribute name="bind" type="token" use="optional" />
  <attribute name="unbind" type="token" use="optional" />
  <attribute name="updated" type="token" use="optional" />
  <anyAttribute processContents="lax" />
</attribute>
</complexType>

<simpleType name="Tjava-types">
<restriction base="string">
  <enumeration value="String" />
  <enumeration value="Long" />
  <enumeration value="Double" />
  <enumeration value="Float" />
  <enumeration value="Integer" />
  <enumeration value="Byte" />
  <enumeration value="Character" />
  <enumeration value="Boolean" />
  <enumeration value="Short" />
</restriction>
</simpleType>
SCR must not require component descriptions to specify the elements in the order required by the schema. SCR must allow other orderings since arbitrary orderings of these elements do not affect the meaning of the component description. Only the relative ordering of property, properties and reference elements have meaning for overriding previously set property values.

The schema is also available in digital form from [5] OSGi XML Schemas.

112.12 org.osgi.service.component

Service Component Package Version 1.2.

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.component; version="[1.2,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.component; version="[1.2,1.3)"
```

112.12.1 Summary

- ComponentConstants - Defines standard names for Service Component constants.
• ComponentContext - A Component Context object is used by a component instance to interact with its execution context including locating services by reference name.

• ComponentException - Unchecked exception which may be thrown by the Service Component Runtime.

• ComponentFactory - When a component is declared with the factory attribute on its component element, the Service Component Runtime will register a Component Factory service to allow new component configurations to be created and activated rather than automatically creating and activating component configuration as necessary.

• ComponentInstance - A ComponentInstance encapsulates a component instance of an activated component configuration.

112.12.2 public interface ComponentConstants

Defines standard names for Service Component constants.

Provider Type Consumers of this API must not implement this type

112.12.2.1 public static final String COMPONENT_FACTORY = "component.factory"

A service registration property for a Component Factory that contains the value of the factory attribute. The value of this property must be of type String.

112.12.2.2 public static final String COMPONENT_ID = "component.id"

A component property that contains the generated id for a component configuration. The value of this property must be of type Long.

The value of this property is assigned by the Service Component Runtime when a component configuration is created. The Service Component Runtime assigns a unique value that is larger than all previously assigned values since the Service Component Runtime was started. These values are NOT persistent across restarts of the Service Component Runtime.

112.12.2.3 public static final String COMPONENT_NAME = "component.name"

A component property for a component configuration that contains the name of the component as specified in the name attribute of the component element. The value of this property must be of type String.

112.12.2.4 public static final int DEACTIVATION_REASON_BUNDLE_STOPPED = 6

The component configuration was deactivated because the bundle was stopped.

Since 1.1

112.12.2.5 public static final int DEACTIVATION_REASON_CONFIGURATION_DELETED = 4

The component configuration was deactivated because its configuration was deleted.

Since 1.1

112.12.2.6 public static final int DEACTIVATION_REASON_CONFIGURATION_MODIFIED = 3

The component configuration was deactivated because its configuration was changed.

Since 1.1

112.12.2.7 public static final int DEACTIVATION_REASON_DISABLED = 1

The component configuration was deactivated because the component was disabled.

Since 1.1

112.12.2.8 public static final int DEACTIVATION_REASON_DISPOSED = 5

The component configuration was deactivated because the component was disposed.
Since 1.1

112.12.9  
```java
public static final int DEACTIVATION_REASON_REFERENCE = 2
```

The component configuration was deactivated because a reference became unsatisfied.

Since 1.1

112.12.10  
```java
public static final int DEACTIVATION_REASON_UNSPECIFIED = 0
```

The reason the component configuration was deactivated is unspecified.

Since 1.1

112.12.11  
```java
public static final String REFERENCE_TARGET_SUFFIX = "\.target"
```

The suffix for reference target properties. These properties contain the filter to select the target services for a reference. The value of this property must be of type String.

112.12.12  
```java
public static final String SERVICE_COMPONENT = "Service-Component"
```

Manifest header specifying the XML documents within a bundle that contain the bundle's Service Component descriptions.

The attribute value may be retrieved from the Dictionary object returned by the Bundle.getHeaders method.

112.12.3  
**public interface ComponentContext**

A Component Context object is used by a component instance to interact with its execution context including locating services by reference name. Each component instance has a unique Component Context.

A component instance may have an activate method. If a component instance has a suitable and accessible activate method, this method will be called when a component configuration is activated. If the activate method takes a ComponentContext argument, it will be passed the component instance's Component Context object. If the activate method takes a BundleContext argument, it will be passed the component instance's Bundle Context object. If the activate method takes a Map argument, it will be passed an unmodifiable Map containing the component properties.

A component instance may have a deactivate method. If a component instance has a suitable and accessible deactivate method, this method will be called when the component configuration is deactivated. If the deactivate method takes a ComponentContext argument, it will be passed the component instance's Component Context object. If the deactivate method takes a BundleContext argument, it will be passed the component instance's Bundle Context object. If the deactivate method takes a Map argument, it will be passed an unmodifiable Map containing the component properties. If the deactivate method takes an int or Integer argument, it will be passed the reason code for the component instance's deactivation.

**Concurrency**  Thread-safe

**Provider Type**  Consumers of this API must not implement this type

112.12.3.1  
```java
public void disableComponent(String name)
```

*name*  
The name of a component.

- Enables the specified component name. The specified component name must be in the same bundle as this component.

112.12.3.2  
```java
public void enableComponent(String name)
```

*name*  
The name of a component or null to indicate all components in the bundle.

- Enables the specified component name. The specified component name must be in the same bundle as this component.
112.12.3.3 public BundleContext getBundleContext()

□ Returns the BundleContext of the bundle which contains this component.

Returns The BundleContext of the bundle containing this component.

112.12.3.4 public ComponentInstance getComponentInstance()

□ Returns the Component Instance object for the component instance associated with this Component Context.

Returns The Component Instance object for the component instance.

112.12.3.5 public Dictionary<String, Object> getProperties()

□ Returns the component properties for this Component Context.

Returns The properties for this Component Context. The Dictionary is read only and cannot be modified.

112.12.3.6 public ServiceReference<?> getServiceReference()

□ If the component instance is registered as a service using the service element, then this method returns the service reference of the service provided by this component instance.

This method will return null if the component instance is not registered as a service.

Returns The ServiceReference object for the component instance or null if the component instance is not registered as a service.

112.12.3.7 public Bundle getUsingBundle()

□ If the component instance is registered as a service using the servicefactory="true" attribute, then this method returns the bundle using the service provided by the component instance.

This method will return null if:

• The component instance is not a service, then no bundle can be using it as a service.
• The component instance is a service but did not specify the servicefactory="true" attribute, then all bundles using the service provided by the component instance will share the same component instance.
• The service provided by the component instance is not currently being used by any bundle.

Returns The bundle using the component instance as a service or null.

112.12.3.8 public Object locateService(String name)

name The name of a reference as specified in a reference element in this component's description.

□ Returns the service object for the specified reference name.

If the cardinality of the reference is 0..n or 1..n and multiple services are bound to the reference, the service with the highest ranking (as specified in its Constants.SERVICE_RANKING property) is returned. If there is a tie in ranking, the service with the lowest service id (as specified in its Constants.SERVICE_ID property); that is, the service that was registered first is returned.

Returns A service object for the referenced service or null if the reference cardinality is 0..1 or 0..n and no bound service is available.

Throws ComponentException– If the Service Component Runtime catches an exception while activating the bound service.

112.12.3.9 public Object locateService(String name, ServiceReference<?> reference)

name The name of a reference as specified in a reference element in this component's description.

reference The ServiceReference to a bound service. This must be a ServiceReference provided to the component via the bind or unbind method for the specified reference name.
112.12.3.10 public Object[] locateServices(String name)

name The name of a reference as specified in a reference element in this component's description.

Returns The service objects for the specified reference name.

Throws ComponentException – If the Service Component Runtime catches an exception while activating the bound service.

112.12.4 public class ComponentException
extends RuntimeException

Unchecked exception which may be thrown by the Service Component Runtime.

112.12.4.1 public ComponentException(String message, Throwable cause)

message The message for the exception.

cause The cause of the exception. May be null.

Returns Constructs a new ComponentException with the specified message and cause.

112.12.4.2 public ComponentException(String message)

message The message for the exception.

Returns Constructs a new ComponentException with the specified message.

112.12.4.3 public ComponentException(Throwable cause)

cause The cause of the exception. May be null.

Returns Constructs a new ComponentException with the specified cause.

112.12.4.4 public Throwable getCause()

Returns The cause of this exception or null if no cause was set.

112.12.4.5 public Throwable initCause(Throwable cause)

cause The cause of this exception.

Returns Initializes the cause of this exception to the specified value.

Throws IllegalArgumentException – If the specified cause is this exception.
IllegalStateException – If the cause of this exception has already been set.

112.12.5 public interface ComponentFactory

When a component is declared with the factory attribute on its component element, the Service Component Runtime will register a Component Factory service to allow new component configura-
tions to be created and activated rather than automatically creating and activating component configuration as necessary.

Concurrence Thread-safe

Provider Type Consumers of this API must not implement this type

112.12.5.1 public ComponentInstance newInstance(Dictionary<String,?> properties)

properties Additional properties for the component configuration or null if there are no additional properties.

□ Create and activate a new component configuration. Additional properties may be provided for the component configuration.

Returns A ComponentInstance object encapsulating the component instance of the component configuration. The component configuration has been activated and, if the component specifies a service element, the component instance has been registered as a service.

 Throws ComponentException– If the Service Component Runtime is unable to activate the component configuration.

112.12.6 public interface ComponentInstance

A ComponentInstance encapsulates a component instance of an activated component configuration. ComponentInstances are created whenever a component configuration is activated.

ComponentInstances are never reused. A new ComponentInstance object will be created when the component configuration is activated again.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

112.12.6.1 public void dispose()

□ Dispose of the component configuration for this component instance. The component configuration will be deactivated. If the component configuration has already been deactivated, this method does nothing.

112.12.6.2 public Object getInstance()

□ Returns the component instance of the activated component configuration.

Returns The component instance or null if the component configuration has been deactivated.

112.13 org.osgi.service.component.annotations

Service Component Annotations Package Version 1.2.

This package is not used at runtime. Annotated classes are processed by tools to generate Component Descriptions which are used at runtime.

112.13.1 Summary

• Activate - Identify the annotated method as the activate method of a Service Component.
• Component - Identify the annotated class as a Service Component.
• ConfigurationPolicy - Configuration Policy for the Component annotation.
• Deactivate - Identify the annotated method as the deactivate method of a Service Component.
• Modified - Identify the annotated method as the modified method of a Service Component.
• Reference - Identify the annotated method as a bind method of a Service Component.
• ReferenceCardinality - Cardinality for the Reference annotation.
• ReferencePolicy - Policy for the Reference annotation.
• ReferencePolicyOption - Policy option for the Reference annotation.

### 112.13.2 @Activate

Identify the annotated method as the activate method of a Service Component.

The annotated method is the activate method of the Component.

This annotation is not processed at runtime by a Service Component Runtime implementation. It must be processed by tools and used to add a Component Description to the bundle.

*See Also* The activate attribute of the component element of a Component Description.

*Since* 1.1

*Retention* CLASS

*Target* METHOD

### 112.13.3 @Component

Identify the annotated class as a Service Component.

The annotated class is the implementation class of the Component.

This annotation is not processed at runtime by a Service Component Runtime implementation. It must be processed by tools and used to add a Component Description to the bundle.

*See Also* The component element of a Component Description.

*Retention* CLASS

*Target* TYPE

#### 112.13.3.1 String name default ""

- The name of this Component.

  If not specified, the name of this Component is the fully qualified type name of the class being annotated.

*See Also* The name attribute of the component element of a Component Description.

#### 112.13.3.2 Class<?>[] service default {}

- The types under which to register this Component as a service.

  If no service should be registered, the empty value `{}` must be specified.

  If not specified, the service types for this Component are all the directly implemented interfaces of the class being annotated.

*See Also* The service element of a Component Description.

#### 112.13.3.3 String factory default ""

- The factory identifier of this Component. Specifying a factory identifier makes this Component a Factory Component.

  If not specified, the default is that this Component is not a Factory Component.

*See Also* The factory attribute of the component element of a Component Description.

#### 112.13.3.4 boolean servicefactory default false

- Declares whether this Component uses the OSGi ServiceFactory concept and each bundle using this Component’s service will receive a different component instance.
If true, this Component uses the OSGi ServiceFactory concept. If false or not specified, this Component does not use the OSGi ServiceFactory concept.

See Also
The servicefactory attribute of the service element of a Component Description.

112.13.3.5

**boolean enabled default true**

Drag

Declares whether this Component is enabled when the bundle containing it is started.

If true or not specified, this Component is enabled. If false, this Component is disabled.

See Also
The enabled attribute of the component element of a Component Description.

112.13.3.6

**boolean immediate default false**

Drag

Declares whether this Component must be immediately activated upon becoming satisfied or whether activation should be delayed.

If true, this Component must be immediately activated upon becoming satisfied. If false, activation of this Component is delayed. If this property is specified, its value must be false if the factory property is also specified or must be true if the service property is specified with an empty value.

If not specified, the default is false if the factory property is specified or the service property is not specified or specified with a non-empty value and true otherwise.

See Also
The immediate attribute of the component element of a Component Description.

112.13.3.7

**String[] property default {}**

Drag

Properties for this Component.

Each property string is specified as "key=value". The type of the property value can be specified in the key as key:type=value. The type must be one of the property types supported by the type attribute of the property element of a Component Description.

To specify a property with multiple values, use multiple key, value pairs. For example, "foo=bar", "foo=baz".

See Also
The property element of a Component Description.

112.13.3.8

**String[] properties default {}**

Drag

Property entries for this Component.

Specifies the name of an entry in the bundle whose contents conform to a standard Java Properties File. The entry is read and processed to obtain the properties and their values.

See Also
The properties element of a Component Description.

112.13.3.9

**String xmlns default ""**

Drag

The XML name space of the Component Description for this Component.

If not specified, the XML name space of the Component Description for this Component should be the lowest Declarative Services XML name space which supports all the specification features used by this Component.

See Also
The XML name space specified for a Component Description.

112.13.3.10

**ConfigurationPolicy configurationPolicy default ConfigurationPolicy.OPTIONAL**

Drag

The configuration policy of this Component.

Controls whether component configurations must be satisfied depending on the presence of a corresponding Configuration object in the OSGi Configuration Admin service. A corresponding configuration is a Configuration object where the PID equals the name of the component.

If not specified, the OPTIONAL configuration policy is used.
String configurationPid default ""

- The configuration PID for the configuration of this Component.
- Allows the configuration PID for this Component to be different than the name of this Component.
- If not specified, the name of this Component is used as the configuration PID of this Component.

See Also The configuration-pid attribute of the component element of a Component Description.

Since 1.2

enumer ConfigurationPolicy

Configuration Policy for the Component annotation.

- Controls whether component configurations must be satisfied depending on the presence of a corresponding Configuration object in the OSGi Configuration Admin service. A corresponding configuration is a Configuration object where the PID is the name of the component.

Since 1.1

OPTIONAL

- Use the corresponding Configuration object if present but allow the component to be satisfied even if the corresponding Configuration object is not present.

REQUIRE

- There must be a corresponding Configuration object for the component configuration to become satisfied.

IGNORE

- Always allow the component configuration to be satisfied and do not use the corresponding Configuration object even if it is present.

@Deactivate

- Identify the annotated method as the deactivate method of a Service Component.
- The annotated method is the deactivate method of the Component.
- This annotation is not processed at runtime by a Service Component Runtime implementation. It must be processed by tools and used to add a Component Description to the bundle.

See Also The deactivate attribute of the component element of a Component Description.

Since 1.1

Retention CLASS

Target METHOD

@Modified

- Identify the annotated method as the modified method of a Service Component.
- The annotated method is the modified method of the Component.
- This annotation is not processed at runtime by a Service Component Runtime implementation. It must be processed by tools and used to add a Component Description to the bundle.

See Also The modified attribute of the component element of a Component Description.
Since 1.1
Retention CLASS
Target METHOD

112.13.7 @Reference

Identify the annotated method as a bind method of a Service Component.
The annotated method is a bind method of the Component.
This annotation is not processed at runtime by a Service Component Runtime implementation. It must be processed by tools and used to add a Component Description to the bundle.
In the generated Component Description for a component, the references must be ordered in ascending lexicographical order (using String.compareTo) of the reference names.

See Also The reference element of a Component Description.

Retention CLASS
Target METHOD

112.13.7.1 String name default ""

- The name of this reference.
  If not specified, the name of this reference is based upon the name of the method being annotated. If the method name begins with bind, set or add, that is removed.

See Also The name attribute of the reference element of a Component Description.

112.13.7.2 Class<?> service default Object.class

- The type of the service to bind to this reference.
  If not specified, the type of the service to bind is based upon the type of the first argument of the method being annotated.

See Also The interface attribute of the reference element of a Component Description.

112.13.7.3 ReferenceCardinality cardinality default ReferenceCardinality.MANDATORY

- The cardinality of the reference.
  If not specified, the reference has a 1..1 cardinality.

See Also The cardinality attribute of the reference element of a Component Description.

112.13.7.4 ReferencePolicy policy default ReferencePolicy.STATIC

- The policy for the reference.
  If not specified, the STATIC reference policy is used.

See Also The policy attribute of the reference element of a Component Description.

112.13.7.5 String target default ""

- The target filter for the reference.

See Also The target attribute of the reference element of a Component Description.

112.13.7.6 String unbind default ""

- The name of the unbind method which is associated with the annotated bind method.
  To declare no unbind method, the value "-" must be used.
If not specified, the name of the unbind method is derived from the name of the annotated bind method. If the annotated method name begins with bind, set or add, that is replaced with unbind, unset or remove, respectively, to derive the unbind method name. Otherwise, un is prefixed to the annotated method name to derive the unbind method name. The unbind method is only set if the component type contains a method with the derived name.

See Also  The unbind attribute of the reference element of a Component Description.

112.13.7.7   ReferencePolicyOption policyOption default ReferencePolicyOption.RELUCTANT

□  The policy option for the reference.
If not specified, the RELUCTANT reference policy option is used.

See Also  The policy-option attribute of the reference element of a Component Description.
Since  1.2

112.13.7.8   String updated default ""

□  The name of the updated method which is associated with the annotated bind method.
To declare no updated method, the value "-" must be used.
If not specified, the name of the updated method is derived from the name of the annotated bind method. If the annotated method name begins with bind, set or add, that is replaced with updated to derive the updated method name. Otherwise, updated is prefixed to the annotated method name to derive the updated method name. The updated method is only set if the component type contains a method with the derived name.

See Also  The updated attribute of the reference element of a Component Description.
Since  1.2

112.13.8   enum ReferenceCardinality
Cardinality for the Reference annotation.
Specifies if the reference is optional and if the component implementation support a single bound service or multiple bound services.

112.13.8.1  OPTIONAL
The reference is optional and unary. That is, the reference has a cardinality of 0..1.

112.13.8.2  MANDATORY
The reference is mandatory and unary. That is, the reference has a cardinality of 1..1.

112.13.8.3  MULTIPLE
The reference is optional and multiple. That is, the reference has a cardinality of 0..n.

112.13.8.4  AT_LEAST_ONE
The reference is mandatory and multiple. That is, the reference has a cardinality of 1..n.

112.13.9   enum ReferencePolicy
Policy for the Reference annotation.

112.13.9.1  STATIC
The static policy is the most simple policy and is the default policy. A component instance never sees any of the dynamics. Component configurations are deactivated before any bound service for a reference having a static policy becomes unavailable. If a target service is available to replace the
bound service which became unavailable, the component configuration must be reactivated and bound to the replacement service.

112.13.9.2 DYNAMIC

The dynamic policy is slightly more complex since the component implementation must properly handle changes in the set of bound services. With the dynamic policy, SCR can change the set of bound services without deactivating a component configuration. If the component uses the event strategy to access services, then the component instance will be notified of changes in the set of bound services by calls to the bind and unbind methods.

112.13.10 enum ReferencePolicyOption

Policy option for the Reference annotation.

Since 1.2

112.13.10.1 RELUCTANT

The reluctant policy option is the default policy option for both static and dynamic reference policies. When a new target service for a reference becomes available, references having the reluctant policy option for the static policy or the dynamic policy with a unary cardinality will ignore the new target service. References having the dynamic policy with a multiple cardinality will bind the new target service.

112.13.10.2 GREEDY

The greedy policy option is a valid policy option for both static and dynamic reference policies. When a new target service for a reference becomes available, references having the greedy policy option will bind the new target service.

112.14 References

[1] Automating Service Dependency Management in a Service-Oriented Component Model
Humberto Cervantes, Richard S. Hall, Proceedings of the Sixth Component-Based Software Engineering Workshop, May 2003, pp. 91-96

[2] Service Binder
Humberto Cervantes, Richard S. Hall
http://gravity.sourceforge.net/servicebinder

http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Properties.html

[4] Extensible Markup Language (XML) 1.0
http://www.w3.org/TR/REC-xml/

[5] OSGi XML Schemas
http://www.osgi.org/Specifications

112.15 Changes

- Configuration Changes on page 198 has been rewritten to more accurately state the effects of configuration changes on component configurations.
**113 Event Admin Service Specification**

**Version 1.3**

**113.1 Introduction**

Nearly all the bundles in an OSGi framework must deal with events, either as an event publisher or as an event handler. So far, the preferred mechanism to disperse those events have been the service interface mechanism.

Dispatching events for a design related to X, usually involves a service of type XListener. However, this model does not scale well for fine grained events that must be dispatched to many different handlers. Additionally, the dynamic nature of the OSGi environment introduces several complexities because both event publishers and event handlers can appear and disappear at any time.

The Event Admin service provides an inter-bundle communication mechanism. It is based on a event publish and subscribe model, popular in many message based systems.

This specification defines the details for the participants in this event model.

**113.1.1 Essentials**

- **Simplifications** - The model must significantly simplify the process of programming an event source and an event handler.
- **Dependencies** - Handle the myriad of dependencies between event sources and event handlers for proper cleanup.
- **Synchronicity** - It must be possible to deliver events asynchronously or synchronously with the caller.
- **Event Window** - Only event handlers that are active when an event is published must receive this event, handlers that register later must not see the event.
- **Performance** - The event mechanism must impose minimal overhead in delivering events.
- **Selectivity** - Event listeners must only receive notifications for the event types for which they are interested.
- **Reliability** - The Event Admin must ensure that events continue to be delivered regardless the quality of the event handlers.
- **Security** - Publishing and receiving events are sensitive operations that must be protected per event type.
- **Extensibility** - It must be possible to define new event types with their own data types.
- **Native Code** - Events must be able to be passed to native code or come from native code.
- **OSGi Events** - The OSGi Framework, as well as a number of OSGi services, already have number of its own events defined. For uniformity of processing, these have to be mapped into generic event types.

**113.1.2 Entities**

- **Event** - An Event object has a topic and a Dictionary object that contains the event properties. It is an immutable object.
- **Event Admin** - The service that provides the publish and subscribe model to Event Handlers and Event Publishers.
- **Event Handler** - A service that receives and handles Event objects.
- **Event Publisher** - A bundle that sends event through the Event Admin service.
- **Event Subscriber** - Another name for an Event Handler.
- **Topic** - The name of an Event type.
- **Event Properties** - The set of properties that is associated with an Event.

**Figure 113.1** The Event Admin service org.osgi.service.event package

![Event Admin Service Diagram](image)

### 113.1.3 Synopsis

The Event Admin service provides a place for bundles to publish events, regardless of their destination. It is also used by Event Handlers to subscribe to specific types of events.

Events are published under a topic, together with a number of event properties. Event Handlers can specify a filter to control the Events they receive on a very fine grained basis.

### 113.1.4 What To Read

- **Architects** - The Event Admin Architecture on page 218 provides an overview of the Event Admin service.
- **Event Publishers** - The Event Publisher on page 221 provides an introduction of how to write an Event Publisher. The Event Admin Architecture on page 218 provides a good overview of the design.
- **Event Subscribers/Handlers** - The Event Handler on page 220 provides the rules on how to subscribe and handle events.

### 113.2 Event Admin Architecture

The Event Admin is based on the Publish-Subscribe pattern. This pattern decouples sources from their handlers by interposing an event channel between them. The publisher posts events to the channel, which identifies which handlers need to be notified and then takes care of the notification process. This model is depicted in Figure 113.2.
In this model, the event source and event handler are completely decoupled because neither has any direct knowledge of the other. The complicated logic of monitoring changes in the event publishers and event handlers is completely contained within the event channel. This is highly advantageous in an OSGi environment because it simplifies the process of both sending and receiving events.

113.3 The Event

Events have the following attributes:

- **Topic** - A topic that defines what happened. For example, when a bundle is started an event is published that has a topic of `org/osgi/framework/BundleEvent/STARTED`.
- **Properties** - Zero or more properties that contain additional information about the event. For example, the previous example event has a property of `bundle.id` which is set to a `Long` object, among other properties.

113.3.1 Topics

The topic of an event defines the *type* of the event. It is fairly granular in order to give handlers the opportunity to register for just the events they are interested in. When a topic is designed, its name should not include any other information, such as the publisher of the event or the data associated with the event, those parts are intended to be stored in the event properties.

The topic is intended to serve as a first-level filter for determining which handlers should receive the event. Event Admin service implementations use the structure of the topic to optimize the dispatching of the events to the handlers.

Topics are arranged in a hierarchical namespace. Each level is defined by a token and levels are separated by solidi (`/`). More precisely, the topic must conform to the following grammar:

\[
\text{topic} ::= \text{token} \ (\ / \ \text{token} \ ) ^ * \\
\]

// See General Syntax Definitions in Core

Topics should be designed to become more specific when going from left to right. Handlers can provide a prefix that matches a topic, using the preferred order allows a handler to minimize the number of prefixes it needs to register.

Topics are case-sensitive. As a convention, topics should follow the reverse domain name scheme used by Java packages to guarantee uniqueness. The separator must be a solidus (`/`) instead of the full stop (`.`).

This specification uses the convention `fully/qualified/package/ClassName/ACTION`. If necessary, a pseudo-class-name is used.

113.3.2 Properties

Information about the actual event is provided as properties. The property name is a case-sensitive string and the value can be any object. Although any Java object can be used as a property value, only `String` objects and the eight primitive types (plus their wrappers) should be used. Other types cannot be passed to handlers that reside external from the Java VM.
Another reason that arbitrary classes should not be used is the mutability of objects. If the values are not immutable, then any handler that receives the event could change the value. Any handlers that received the event subsequently would see the altered value and not the value as it was when the event was sent.

The topic of the event is available as a property with the key `EVENT_TOPIC`. This allows filters to include the topic as a condition if necessary.

### 113.3.3 High Performance

An event processing system can become a bottleneck in large systems. One expensive aspect of the Event object is its properties and its immutability. This combination requires the Event object to create a copy of the properties for each object. There are many situations where the same properties are dispatched through Event Admin, the topic is then used to signal the information. Creating the copy of the properties can therefore take unnecessary CPU time and memory. However, the immutability of the Event object requires the properties to be immutable.

For this reason, this specification also provides an immutable Map with the Event Properties class. This class implements an immutable map that is recognized and trusted by the Event object to not mutate. Using an Event Properties object allows a client to create many different Event objects with different topics but sharing the same properties object.

The following example shows how an event poster can limit the copying of the properties.

```java
void foo(EventAdmin eventAdmin) {
    Map<String, Object> props = new HashMap<String, Object>();
    props.put("foo", 1);
    EventProperties eventProps = new EventProperties(props);

    for (int i=0; i<1000; i++)
        eventAdmin.postEvent(new Event("my/topic/" + i, eventProps));
}
```

### 113.4 Event Handler

Event handlers must be registered as services with the OSGi framework under the object class `org.osgi.service.event.EventHandler`.

Event handlers should be registered with a property (constant from the `EventConstants` class) `EVENT_TOPIC`. The value being a `String`, `String[]` or `Collection<String>` object that describes which topics the handler is interested in. A wildcard asterisk (`'*'`) may be used as the last token of a topic name, for example `com/action/*`. This matches any topic that shares the same first tokens. For example, `com/action/*` matches `com/action/listen`.

Event Handlers which have not specified the `EVENT_TOPIC` service property must not receive events.

The value of each entry in the `EVENT_TOPIC` service registration property must conform to the following grammar:

```
topic-scope ::= '*' | ( topic '/*' )
```

Event handlers can also be registered with a service property named `EVENT_FILTER`. The value of this property must be a string containing a Framework filter specification. Any of the event's properties can be used in the filter expression.

```
event-filter ::= filter  // See Filter Syntax in Core
```
Each Event Handler is notified for any event which belongs to the topics the handler has expressed an interest in. If the handler has defined a `EVENT_FILTER` service property then the event properties must also match the filter expression. If the filter is an error, then the Event Admin service should log a warning and further ignore the Event Handler.

For example, a bundle wants to see all Log Service events with a level of `WARNING` or `ERROR`, but it must ignore the `INFO` and `DEBUG` events. Additionally, the only events of interest are when the bundle symbolic name starts with `com.acme`.

```java
public AcmeWatchDog implements BundleActivator, EventHandler {
    final static String[] topics = new String[] {
        "org/osgi/service/log/LogEntry/LOG_WARNING",
        "org/osgi/service/log/LogEntry/LOG_ERROR"
    };

    public void start(BundleContext context) {
        Dictionary d = new Hashtable();
        d.put(EventConstants.EVENT_TOPIC, topics );
        d.put(EventConstants.EVENT_FILTER,
            "(bundle.symbolicName=com.acme.*)");
        context.registerService( EventHandler.class.getName(),
            this, d );
    }

    public void stop(BundleContext context) {};

    public void handleEvent(Event event) {
        //...
    }
}
```

If there are multiple Event Admin services registered with the Framework then all Event Admin services must send their published events to all registered Event Handlers.

### 113.4 Ordering

In the default case, an Event Handler will receive posted (asynchronous) events from a single thread in the same order as they were posted. Maintaining this ordering guarantee requires the Event Admin to serialize the delivery of events instead of, for example, delivering the events on different worker threads. There are many scenarios where this ordering is not really required. For this reason, an Event Handler can signal to the Event Admin that events can be delivered out of order. This is notified with the `EVENT_DELIVERY` service property. This service property can be used in the following way:

- Not set or set to both - The Event Admin must deliver the events in the proper order.
- `DELIVERY_ASYNC_ORDERED` - Events must be delivered in order.
- `DELIVERY_ASYNC_UNORDERED` - Allow the events to be delivered in any order.

### 113.5 Event Publisher

To fire an event, the event source must retrieve the Event Admin service from the OSGi service registry. Then it creates the event object and calls one of the Event Admin service's methods to fire the event either synchronously or asynchronously.

The following example is a class that publishes a time event every 60 seconds.

```java
public class TimerEvent extends Thread
```
implements BundleActivator {
    Hashtable time = new Hashtable();
    ServiceTracker tracker;

    public TimerEvent() { super("TimerEvent"); }

    public void start(BundleContext context) {
        tracker = new ServiceTracker(context,
            EventAdmin.class.getName(), null);
        tracker.open();
        start();
    }

    public void stop(BundleContext context) {
        interrupt();
        tracker.close();
    }

    public void run() {
        while (! Thread.interrupted()) try {
            Calendar c = Calendar.getInstance();
            set(c, Calendar.MINUTE, "minutes");
            set(c, Calendar.HOUR, "hours");
            set(c, Calendar.DAY_OF_MONTH, "day");
            set(c, Calendar.MONTH, "month");
            set(c, Calendar.YEAR, "year");

            EventAdmin ea = (EventAdmin) tracker.getService();
            if ( ea != null )
                ea.sendEvent(new Event("com/acme/timer",
                    time));
            Thread.sleep(60000-c.get(Calendar.SECOND)*1000);
        } catch( InterruptedException e ) {
            return;
        }

        void set( Calendar c, int field, String key ) {
            time.put( key, new Integer(c.get(field)));
        }
    }
}
### 113.6 Specific Events

#### 113.6.1 General Conventions

Some handlers are more interested in the contents of an event rather than what actually happened. For example, a handler wants to be notified whenever an Exception is thrown anywhere in the system. Both Framework Events and Log Entry events may contain an exception that would be of interest to this hypothetical handler. If both Framework Events and Log Entries use the same property names then the handler can access the Exception in exactly the same way. If some future event type follows the same conventions then the handler can receive and process the new event type even though it had no knowledge of it when it was compiled.

The following properties are suggested as conventions. When new event types are defined they should use these names with the corresponding types and values where appropriate. These values should be set only if they are not null.

A list of these property names can be found in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUNDLE_SIGNER</td>
<td>String</td>
<td>Collection</td>
</tr>
<tr>
<td>BUNDLE_VERSION</td>
<td>Version</td>
<td>A bundle’s version</td>
</tr>
<tr>
<td>BUNDLE_SYMBOLICNAME</td>
<td>String</td>
<td>A bundle’s symbolic name</td>
</tr>
<tr>
<td>EVENT</td>
<td>Object</td>
<td>The actual event object. Used when rebroadcasting an event that was sent via some other event mechanism</td>
</tr>
<tr>
<td>EXCEPTION</td>
<td>Throwable</td>
<td>An exception or error</td>
</tr>
<tr>
<td>EXCEPTION_MESSAGE</td>
<td>String</td>
<td>Must be equal to exception.getMessage().</td>
</tr>
<tr>
<td>EXCEPTION_CLASS</td>
<td>String</td>
<td>Must be equal to the name of the Exception class.</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>String</td>
<td>A human-readable message that is usually not localized.</td>
</tr>
<tr>
<td>SERVICE</td>
<td>Service Reference</td>
<td>A Service Reference</td>
</tr>
<tr>
<td>SERVICE_ID</td>
<td>Long</td>
<td>A service’s id</td>
</tr>
<tr>
<td>SERVICE_OBJECTCLASS</td>
<td>String[]</td>
<td>A service’s objectClass</td>
</tr>
<tr>
<td>SERVICE_PID</td>
<td>String</td>
<td>Collection</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Long</td>
<td>The time when the event occurred, as reported by System.currentTimeMillis()</td>
</tr>
</tbody>
</table>

The topic of an OSGi event is constructed by taking the fully qualified name of the event class, substituting a solidus (‘/’) for every full stop, and appending a solidus followed by the name of the constant that defines the event type. For example, the topic of

```
BundleEvent.STARTED
```

becomes

```
org.osgi.framework.BundleEvent/STARTED
```

If a type code for the event is unknown then the event must be ignored.
113.6.2 OSGi Events

In order to present a consistent view of all the events occurring in the system, the existing Framework-level events are mapped to the Event Admin's publish-subscribe model. This allows event subscribers to treat framework events exactly the same as other events.

It is the responsibility of the Event Admin service implementation to map these Framework events to its queue.

The properties associated with the event depends on its class as outlined in the following sections.

113.6.3 Framework Event

Framework Events must be delivered asynchronously with a topic of:

org.osgi.framework/FrameworkEvent/<eventtype>

The following event types are supported:

STARTED
ERROR
PACKAGES_REFRESHED
STARTLEVEL_CHANGED
WARNING
INFO

Other events are ignored, no event will be send by the Event Admin. The following event properties must be set for a Framework Event.

- event - (FrameworkEvent) The original event object.

If the FrameworkEvent getBundle method returns a non-null value, the following fields must be set:

- bundle.id - (Long) The source's bundle id.
- bundle.symbolicName - (String) The source bundle's symbolic name. Only set if the bundle's symbolic name is not null.
- bundle.version - (Version) The version of the bundle, if set.
- bundle.signer - (String|Collection<String>) The DNs of the signers.
- bundle - (Bundle) The source bundle.

If the FrameworkEvent getThrowable method returns a non-null value:

- exception.class - (String) The fully-qualified class name of the attached Exception.
- exception.message - (String) The message of the attached exception. Only set if the Exception message is not null.
- exception - (Throwable) The Exception returned by the getThrowable method.

113.6.4 Bundle Event

Framework Events must be delivered asynchronously with a topic of:

org.osgi.framework/BundleEvent/<event type>

The following event types are supported:

INSTALLED
STARTED
STOPPED
Unknown events must be ignored.

The following event properties must be set for a Bundle Event. If listeners require synchronous delivery then they should register a Synchronous Bundle Listener with the Framework.

- `event` - (BundleEvent) The original event object.
- `bundle.id` - (Long) The source's bundle id.
- `bundle.symbolicName` - (String) The source bundle's symbolic name. Only set if the bundle's symbolic name is not null.
- `bundle.version` - (Version) The version of the bundle, if set.
- `bundle.signer` - (String|Collection<String>) The DNs of the signers.
- `bundle` - (Bundle) The source bundle.

### 113.6.5 Service Event

Service Events must be delivered asynchronously with the topic:

`org/osgi/framework/ServiceEvent/<eventtype>`

The following event types are supported:

- REGISTERED
- MODIFIED
- UNREGISTERING

Unknown events must be ignored.

- `event` - (ServiceEvent) The original Service Event object.
- `service` - (ServiceReference) The result of the `getServiceReference` method
- `service.id` - (Long) The service's ID.
- `service.pid` - (String or Collection<String>) The service's persistent identity. Only set if not null. If the PID is specified as a String[], then it must be coerced into a Collection<String>.
- `service.objectClass` - (String[]) The service's object class.

### 113.6.6 Other Event Sources

Several OSGi service specifications define their own event model. It is the responsibility of these services to map their events to Event Admin events. Event Admin is seen as a core service that will be present in most devices. However, if there is no Event Admin service present, applications are not mandated to buffer events.

### 113.7 Event Admin Service

The Event Admin service must be registered as a service with the object class `org.osgi.service.event.EventAdmin`. Multiple Event Admin services can be registered. Publishers should publish their event on the Event Admin service with the highest value for the `SERVICE_RANKING` service property. This is the service selected by the `getServiceReference` method.

The Event Admin service is responsible for tracking the registered handlers, handling event notifications and providing at least one thread for asynchronous event delivery.
113.7.1 Synchronous Event Delivery

Synchronous event delivery is initiated by the `sendEvent` method. When this method is invoked, the Event Admin service determines which handlers must be notified of the event and then notifies each one in turn. The handlers can be notified in the caller’s thread or in an event-delivery thread, depending on the implementation. In either case, all notifications must be completely handled before the `sendEvent` method returns to the caller.

Synchronous event delivery is significantly more expensive than asynchronous delivery. All things considered equal, the asynchronous delivery should be preferred over the synchronous delivery.

Callers of this method will need to be coded defensively and assume that synchronous event notifications could be handled in a separate thread. That entails that they must not be holding any monitors when they invoke the `sendEvent` method. Otherwise they significantly increase the likelihood of deadlocks because Java monitors are not reentrant from another thread by definition. Not holding monitors is good practice even when the event is dispatched in the same thread.

113.7.2 Asynchronous Event Delivery

Asynchronous event delivery is initiated by the `postEvent` method. When this method is invoked, the Event Admin service must determine which handlers are interested in the event. By collecting this list of handlers during the method invocation, the Event Admin service ensures that only handlers that were registered at the time the event was posted will receive the event notification. This is the same as described in Delivering Events of OSGi Core Release 6.

The Event Admin service can use more than one thread to deliver events. If it does then it must guarantee that each handler receives the events in the same order as the events were posted, unless this handler allows unordered deliver, see Ordering on page 221. This ensures that handlers see events in their expected order. For example, for some handlers it would be an error to see a destroyed event before the corresponding created event.

Before notifying each handler, the event delivery thread must ensure that the handler is still registered in the service registry. If it has been unregistered then the handler must not be notified.

113.7.3 Order of Event Delivery

Asynchronous events are delivered in the order in which they arrive in the event queue. Thus if two events are posted by the same thread then they will be delivered in the same order (though other events may come between them). However, if two or more events are posted by different threads then the order in which they arrive in the queue (and therefore the order in which they are delivered) will depend very much on subtle timing issues. The event delivery system cannot make any guarantees in this case. An Event Handler can indicate that the ordering is not relevant, allowing the Event Admin to more aggressively parallelize the event deliver, see Ordering on page 221.

Synchronous events are delivered as soon as they are sent. If two events are sent by the same thread, one after the other, then they must be guaranteed to be processed serially and in the same order. However, if two events are sent by different threads then no guarantees can be made. The events can be processed in parallel or serially, depending on whether or not the Event Admin service dispatches synchronous events in the caller’s thread or in a separate thread.

Note that if the actions of a handler trigger a synchronous event, then the delivery of the first event will be paused and delivery of the second event will begin. Once delivery of the second event has completed, delivery of the first event will resume. Thus some handlers may observe the second event before they observe the first one.
113.8 Reliability

113.8.1 Exceptions in callbacks

If a handler throws an Exception during delivery of an event, it must be caught by the Event Admin service and handled in some implementation specific way. If a Log Service is available the exception should be logged. Once the exception has been caught and dealt with, the event delivery must continue with the next handlers to be notified, if any.

As the Log Service can also forward events through the Event Admin service there is a potential for a loop when an event is reported to the Log Service.

113.8.2 Dealing with Stalled Handlers

Event handlers should not spend too long in the `handleEvent` method. Doing so will prevent other handlers in the system from being notified. If a handler needs to do something that can take a while, it should do it in a different thread.

An event admin implementation can attempt to detect stalled or deadlocked handlers and deal with them appropriately. Exactly how it deals with this situation is left as implementation specific. One allowed implementation is to mark the current event delivery thread as invalid and spawn a new event delivery thread. Event delivery must resume with the next handler to be notified.

Implementations can choose to blacklist any handlers that they determine are misbehaving. Blacklisted handlers must not be notified of any events. If a handler is blacklisted, the event admin should log a message that explains the reason for it.

113.9 Inter-operability with Native Applications

Implementations of the Event Admin service can support passing events to, and/or receiving events from native applications.

If the implementation supports native inter-operability, it must be able to pass the topic of the event and its properties to/from native code. Implementations must be able to support property values of the following types:

- String objects, including full Unicode support
- Integer, Long, Byte, Short, Float, Double, Boolean, Character objects
- Single-dimension arrays of the above types (including String)
- Single-dimension arrays of Java’s eight primitive types (int, long, byte, short, float, double, boolean, char)

Implementations can support additional types. Property values of unsupported types must be silently discarded.

113.10 Security

113.10.1 Topic Permission

The `TopicPermission` class allows fine-grained control over which bundles may post events to a given topic and which bundles may receive those events.

The target parameter for the permission is the topic name. `TopicPermission` classes uses a wildcard matching algorithm similar to the `BasicPermission` class, except that solidi (‘/’ \u002F) are used as
separators instead of full stop characters. For example, a name of a/b/implies a/b/c but not x/y/z or a/b.

There are two available actions: PUBLISH and SUBSCRIBE. These control a bundle's ability to either publish or receive events, respectively. Neither one implies the other.

113.10.2 Required Permissions

Bundles that need to register an event handler must be granted ServicePermission[org.osgi.service.event.EventHandler, REGISTER]. In addition, handlers require TopicPermission[<topic>, SUBSCRIBE] for each topic they want to be notified about.

Bundles that need to publish an event must be granted ServicePermission[org.osgi.service.event.EventAdmin, GET] so that they may retrieve the Event Admin service and use it. In addition, event sources require TopicPermission[<topic>, PUBLISH] for each topic they want to send events to.

Bundles that need to iterate the handlers registered with the system must be granted ServicePermission[org.osgi.service.event.EventHandler, GET] to retrieve the event handlers from the service registry.

Only a bundle that contains an Event Admin service implementation should be granted ServicePermission[org.osgi.service.event.EventAdmin, REGISTER] to register the event channel admin service.

113.10.3 Security Context During Event Callbacks

During an event notification, the Event Admin service's Protection Domain will be on the stack above the handler's Protection Domain. In the case of a synchronous event, the event publisher's protection domain can also be on the stack.

Therefore, if a handler needs to perform a secure operation using its own privileges, it must invoke the doPrivileged method to isolate its security context from that of its caller.

The event delivery mechanism must not wrap event notifications in a doPrivileged call.

113.11 org.osgi.service.event

Event Admin Package Version 1.3.

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.event; version="[1.3,2.0)"

Example import for providers implementing the API in this package:
Import-Package: org.osgi.service.event; version="[1.3,1.4)"

113.11.1 Summary

- Event - An event.
- EventAdmin - The Event Admin service.
- EventConstants - Defines standard names for EventHandler properties.
- EventHandler - Listener for Events.
- EventProperties - The properties for an Event.
• TopicPermission - A bundle's authority to publish or subscribe to event on a topic.

113.11.2 **public class Event**

An event. Event objects are delivered to EventHandler services which subscribe to the topic of the event.

*Concurrency* Immutable

113.11.2.1 **public Event(String topic,Map<String,?> properties)**

*topic* The topic of the event.

*properties* The event's properties (may be null). A property whose key is not of type String will be ignored.

□ Constructs an event.

*Throws* IllegalArgumentException – If topic is not a valid topic name.

*Since* 1.2

113.11.2.2 **public Event(String topic,Dictionary<String,?> properties)**

*topic* The topic of the event.

*properties* The event's properties (may be null). A property whose key is not of type String will be ignored.

□ Constructs an event.

*Throws* IllegalArgumentException – If topic is not a valid topic name.

113.11.2.3 **public final boolean containsProperty(String name)**

*name* The name of the property.

□ Indicate the presence of an event property. The event topic is present using the property name "event.topics".

*Returns* true if a property with the specified name is in the event. This property may have a null value. false otherwise.

*Since* 1.3

113.11.2.4 **public boolean equals(Object object)**

*object* The Event object to be compared.

□ Compares this Event object to another object.

An event is considered to be equal to another event if the topic is equal and the properties are equal. The properties are compared using the java.util.Map.equals() rules which includes identity comparison for array values.

*Returns* true if object is a Event and is equal to this object; false otherwise.

113.11.2.5 **public final Object getProperty(String name)**

*name* The name of the property to retrieve.

□ Retrieve the value of an event property. The event topic may be retrieved with the property name "event.topics".

*Returns* The value of the property, or null if not found.

113.11.2.6 **public final String[] getPropertyNames()**

□ Returns a list of this event's property names. The list will include the event topic property name "event.topics".
Returns A non-empty array with one element per property.

113.11.2.7 public final String getTopic()

- Returns the topic of this event.

Returns The topic of this event.

113.11.2.8 public int hashCode()

- Returns a hash code value for this object.

Returns An integer which is a hash code value for this object.

113.11.2.9 public final boolean matches(Filter filter)

- The filter to test.

filter Tests this event’s properties against the given filter using a case sensitive match.

Returns true If this event’s properties match the filter, false otherwise.

113.11.2.10 public String toString()

- Returns the string representation of this event.

Returns The string representation of this event.

113.11.3 public interface EventAdmin

The Event Admin service. Bundles wishing to publish events must obtain the Event Admin service and call one of the event delivery methods.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

113.11.3.1 public void postEvent(Event event)

- The event to send to all listeners which subscribe to the topic of the event.

event Initiate asynchronous, ordered delivery of an event. This method returns to the caller before delivery of the event is completed. Events are delivered in the order that they are received by this method.

Throws SecurityException – If the caller does not have TopicPermission[topic,PUBLISH] for the topic specified in the event.

113.11.3.2 public void sendEvent(Event event)

- The event to send to all listeners which subscribe to the topic of the event.

event Initiate synchronous delivery of an event. This method does not return to the caller until delivery of the event is completed.

Throws SecurityException – If the caller does not have TopicPermission[topic,PUBLISH] for the topic specified in the event.

113.11.4 public interface EventConstants

Defines standard names for EventHandler properties.

Provider Type Consumers of this API must not implement this type

113.11.4.1 public static final String BUNDLE = "bundle"

The Bundle object of the bundle relevant to the event. The type of the value for this event property is Bundle.
Since 1.1

113.11.4.2 public static final String BUNDLE_ID = "bundle.id"
The Bundle id of the bundle relevant to the event. The type of the value for this event property is Long.

Since 1.1

113.11.4.3 public static final String BUNDLE_SIGNER = "bundle.signer"
The Distinguished Names of the signers of the bundle relevant to the event. The type of the value for this event property is String or Collection of String.

113.11.4.4 public static final String BUNDLE_SYMBOLICNAME = "bundle.symbolicName"
The Bundle Symbolic Name of the bundle relevant to the event. The type of the value for this event property is String.

Since 1.2

113.11.4.5 public static final String BUNDLE_VERSION = "bundle.version"
The version of the bundle relevant to the event. The type of the value for this event property is Version.

Since 1.3

113.11.4.6 public static final String DELIVERY_ASYNC_ORDERED = "async.ordered"
Event Handler delivery quality value specifying the Event Handler requires asynchronously delivered events be delivered in order. Ordered delivery is the default for asynchronously delivered events.

This delivery quality value is mutually exclusive with DELIVERY_ASYNC_UNORDERED. However, if both this value and DELIVERY_ASYNC_UNORDERED are specified for an event handler, this value takes precedence.

See Also EVENT_DELIVERY

Since 1.3

113.11.4.7 public static final String DELIVERY_ASYNC_UNORDERED = "async.unordered"
Event Handler delivery quality value specifying the Event Handler does not require asynchronously delivered events be delivered in order. This may allow an Event Admin implementation to optimize asynchronous event delivery by relaxing ordering requirements.

This delivery quality value is mutually exclusive with DELIVERY_ASYNC_ORDERED. However, if both this value and DELIVERY_ASYNC_ORDERED are specified for an event handler, DELIVERY_ASYNC_ORDERED takes precedence.

See Also EVENT_DELIVERY

Since 1.3

113.11.4.8 public static final String EVENT = "event"
The forwarded event object. Used when rebroadcasting an event that was sent via some other event mechanism. The type of the value for this event property is Object.

113.11.4.9 public static final String EVENT_DELIVERY = "event.delivery"
Service Registration property specifying the delivery qualities requested by an Event Handler service.

Event handlers MAY be registered with this property. Each value of this property is a string specifying a delivery quality for the Event handler.
The value of this property must be of type String, String[], or Collection<String>.

See Also DELIVERY_ASYNC_ORDERED, DELIVERY_ASYNC_UNORDERED

Since 1.3

113.11.4.10 public static final String EVENT_FILTER = "event.filter"

Service Registration property specifying a filter to further select Events of interest to an Event Handler service.

Event handlers MAY be registered with this property. The value of this property is a string containing an LDAP-style filter specification. Any of the event's properties may be used in the filter expression. Each event handler is notified for any event which belongs to the topics in which the handler has expressed an interest. If the event handler is also registered with this service property, then the properties of the event must also match the filter for the event to be delivered to the event handler.

If the filter syntax is invalid, then the Event Handler must be ignored and a warning should be logged.

The value of this property must be of type String.

See Also Event, Filter

113.11.4.11 public static final String EVENT_TOPIC = "event.topics"

Service registration property specifying the Event topics of interest to an Event Handler service.

Event handlers SHOULD be registered with this property. Each value of this property is a string that describe the topics in which the handler is interested. An asterisk ("*") may be used as a trailing wildcard. Event Handlers which do not have a value for this property must not receive events. More precisely, the value of each string must conform to the following grammar:

```
topic-description := '*' | topic ( '/' topic )?
topic := token ( '/' token )*
```

The value of this property must be of type String, String[], or Collection<String>.

See Also Event

113.11.4.12 public static final String EXCEPTION = "exception"

An exception or error. The type of the value for this event property is Throwable.

113.11.4.13 public static final String EXCEPTION_CLASS = "exception.class"

The name of the exception type. Must be equal to the name of the class of the exception in the event property EXCEPTION. The type of the value for this event property is String.

Since 1.1

113.11.4.14 public static final String EXCEPTION_MESSAGE = "exception.message"

The exception message. Must be equal to the result of calling getMessage() on the exception in the event property EXCEPTION. The type of the value for this event property is String.

113.11.4.15 public static final String EXCEPTION_CLASS = "exception.class"

This constant was released with an incorrectly spelled name. It has been replaced by EXCEPTION_CLASS

Deprecated As of 1.1, replaced by EXCEPTION_CLASS

113.11.4.16 public static final String MESSAGE = "message"

A human-readable message that is usually not localized. The type of the value for this event property is String.
public static final String SERVICE = "service"
A service reference. The type of the value for this event property is ServiceReference.

public static final String SERVICE_ID = "service.id"
A service's id. The type of the value for this event property is Long.

public static final String SERVICE_OBJECTCLASS = "service.objectClass"
A service's objectClass. The type of the value for this event property is String[].

public static final String SERVICE_PID = "service.pid"
A service's persistent identity. The type of the value for this event property is String or Collection of String.

public static final String TIMESTAMP = "timestamp"
The time when the event occurred, as reported by System.currentTimeMillis(). The type of the value for this event property is Long.

113.11.5 public interface EventHandler
Listener for Events.
EventHandler objects are registered with the Framework service registry and are notified with an Event object when an event is sent or posted.

EventHandler objects can inspect the received Event object to determine its topic and properties. EventHandler objects must be registered with a service property EventConstants.EVENT_TOPIC whose value is the list of topics in which the event handler is interested.

For example:

String[] topics = new String[] {"com/isv/*"};
Hashtable ht = new Hashtable();
ht.put(EventConstants.EVENT_TOPIC, topics);
context.registerService(EventHandler.class.getName(), this, ht);

Event Handler services can also be registered with an EventConstants.EVENT_FILTER service property to further filter the events. If the syntax of this filter is invalid, then the Event Handler must be ignored by the Event Admin service. The Event Admin service should log a warning.

Security Considerations. Bundles wishing to monitor Event objects will require ServicePermission[EventHandler,REGISTER] to register an EventHandler service. The bundle must also have TopicPermission[topic,SUBSCRIBE] for the topic specified in the event in order to receive the event.

See Also  Event

Concurrence  Thread-safe

113.11.5.1 public void handleEvent(Event event)

event  The event that occurred.

Calling the EventAdmin service to notify the listener of an event.

113.11.6 public class EventProperties
implements Map<String,Object>
The properties for an Event. An event source can create an EventProperties object if it needs to reuse the same event properties for multiple events.
The keys are all of type String. The values are of type Object. The key "event.topics" is ignored as event topics can only be set when an Event is constructed.

Once constructed, an EventProperties object is unmodifiable. However, the values of the map used to construct an EventProperties object are still subject to modification as they are not deeply copied.

Since 1.3

Concurrency Immutable

113.11.6.1 public EventProperties(Map<String,?> properties)

properties The properties to use for this EventProperties object (may be null).

- Create an EventProperties from the specified properties.
  The specified properties will be copied into this EventProperties. Properties whose key is not of type String will be ignored. A property with the key "event.topics" will be ignored.

113.11.6.2 public void clear()

- This method throws UnsupportedOperationException.

Throws UnsupportedOperationException - if called.

113.11.6.3 public boolean containsKey(Object name)

name The property name.

- Indicates if the specified property is present.

Returns true If the property is present, false otherwise.

113.11.6.4 public boolean containsValue(Object value)

value The property value.

- Indicates if the specified value is present.

Returns true If the value is present, false otherwise.

113.11.6.5 public Set<Map.Entry<String,Object>> entrySet()

- Return the property entries.

Returns A set containing the property name/value pairs.

113.11.6.6 public boolean equals(Object object)

object The EventProperties object to be compared.

- Compares this EventProperties object to another object.
  The properties are compared using the java.util.Map.equals() rules which includes identity comparison for array values.

Returns true if object is a EventProperties and is equal to this object; false otherwise.

113.11.6.7 public Object get(Object name)

name The name of the specified property.

- Return the value of the specified property.

Returns The value of the specified property.

113.11.6.8 public int hashCode()

- Returns a hash code value for this object.
Returns An integer which is a hash code value for this object.

113.11.6.9 public boolean isEmpty()
□ Indicate if this properties is empty.
Returns true If this properties is empty, false otherwise.

113.11.6.10 public Set<String> keySet()
□ Return the names of the properties.
Returns The names of the properties.

113.11.6.11 public Object put(String key, Object value)
□ This method throws UnsupportedOperationException.
Throws UnsupportedOperationException— if called.

113.11.6.12 public void putAll(Map<? extends String, ? extends Object> map)
□ This method throws UnsupportedOperationException.
Throws UnsupportedOperationException— if called.

113.11.6.13 public Object remove(Object key)
□ This method throws UnsupportedOperationException.
Throws UnsupportedOperationException— if called.

113.11.6.14 public int size()
□ Return the number of properties.
Returns The number of properties.

113.11.6.15 public String toString()
□ Returns the string representation of this object.
Returns The string representation of this object.

113.11.6.16 public Collection<Object> values()
□ Return the properties values.
Returns The values of the properties.

113.11.7 public final class TopicPermission extends Permission
A bundle's authority to publish or subscribe to event on a topic.
A topic is a slash-separated string that defines a topic.
For example:
org.osgi.service/foo/FooEvent/ACTION
TopicPermission has two actions: publish and subscribe.

Concurrency Thread-safe

113.11.7.1 public static final String PUBLISH = "publish"
The action string publish.
113.11.7.2  public static final String SUBSCRIBE = "subscribe"

The action string subscribe.

113.11.7.3  public TopicPermission(String name, String actions)

name  Topic name.

actions  publish, subscribe (canonical order).

Defines the authority to publish and/or subscribe to a topic within the EventAdmin service.

The name is specified as a slash-separated string. Wildcards may be used. For example:

- org/osgi/service/fooFooEvent/ACTION
- com/isv/*
- *

A bundle that needs to publish events on a topic must have the appropriate TopicPermission for that topic; similarly, a bundle that needs to subscribe to events on a topic must have the appropriate TopicPermission for that topic.

113.11.7.4  public boolean equals(Object obj)

obj  The object to test for equality with this TopicPermission object.

Determines the equality of two TopicPermission objects. This method checks that specified TopicPermission has the same topic name and actions as this TopicPermission object.

Returns  true if obj is a TopicPermission, and has the same topic name and actions as this TopicPermission object; false otherwise.

113.11.7.5  public String getActions()

Returns the canonical string representation of the TopicPermission actions.

Always returns present TopicPermission actions in the following order: publish, subscribe.

Returns  Canonical string representation of the TopicPermission actions.

113.11.7.6  public int hashCode()

Returns the hash code value for this object.

Returns  A hash code value for this object.

113.11.7.7  public boolean implies(Permission p)

p  The target permission to interrogate.

Determines if the specified permission is implied by this object.

This method checks that the topic name of the target is implied by the topic name of this object. The list of TopicPermission actions must either match or allow for the list of the target object to imply the target TopicPermission action.

- x/y/*, "publish" -> x/y/z, "subscribe" is true
- *, "subscribe" -> x/y, "subscribe" is true
- *, "publish" -> x/y, "subscribe" is false
- x/y, "publish" -> x/y/z, "publish" is false

Returns  true if the specified TopicPermission action is implied by this object; false otherwise.
113.11.7.8  public PermissionCollection newPermissionCollection()

□ Returns a new PermissionCollection object suitable for storing TopicPermission objects.

*Returns*  A new PermissionCollection object.
121 Blueprint Container Specification

Version 1.0

121.1 Introduction

One of the great promises of object oriented languages was the greater reuse it should enable. However, over time it turned out that reuse was still hard. One of the key reasons was coupling. Trying to reuse a few classes usually ended up in dragging in many more classes, that in their turn dragged in even more classes, ad nauseum.

One of the key innovations in the Java language to address this coupling issue were interfaces. Interfaces significantly could minimize coupling because they were void of any implementation details. Any class can use an interface, where that interface can be implemented by any other class. However, coupling was still necessary because objects need to be created, and for creating an object its concrete class is necessary.

One of the most successful insights in the software industry of late has been inversion of control, or more specific dependency injection. With dependency injection, an object is given the collaborators that it needs to work with. By not creating these dependencies itself, the object is not coupled to the concrete type of these implementations and their transitive implementation dependencies. However, these objects are not useful on their own, they can only function when an external party provides these objects with their collaborating objects.

An injection framework creates these objects, and also their concrete dependencies, and wires them together. Injection frameworks can significantly increase reuse and provide increased flexibility. For example, during testing it is possible to inject mocked up objects instead of the actual objects.

There exists a number of these injection frameworks in the market, for example [1] Spring Framework, [3] Guice, and [4] Picocontainer. These containers are configured with XML, Java annotations, or provide automatic configuration based on types.

Decoupling is one of the primary drivers for the OSGi specifications. The module layer provides many mechanisms to hide implementation details and explicitly defines any dependencies. The service layer provides a mechanism to collaborate with other bundles without caring about who that other bundle is. However, using the OSGi APIs to construct an application out of services and objects also implies coupling to these OSGi APIs.

This specification therefore defines a dependency injection framework, specifically for OSGi bundles, that understands the unique dynamic nature of services. It provides an OSGi bundle programming model with minimal implementation dependencies and virtually no accidental complexity in the Java code. 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.

This Blueprint Container specification is derived from the [2] Spring Dynamic Modules project.

121.1.1 Essentials

- **Dependency Injection Framework** - Provide an advanced dependency injection framework for bundles that can create and wire objects and services together into an application.
- **Inversion of Control** (IOC) A pattern in which a framework/library provides the control over the component instances instead of the other way around. Dependency injection is a form of IOC.
• **Extender Model** - Enable the configuration of components inside a bundle based on configuration data provided by the bundle developer. The life cycle of these components is controlled by the extender based on the extended bundle’s state.

• **Unencumbered** - Do not require any special bundle activator or other code to be written inside the bundle in order to have components instantiated and configured.

• **Services** - Enable the usage of OSGi services as injected dependencies.

• **Dependencies** - Allow components to depend on other components like services and beans as well as register as services, with the full breadth of the OSGi capabilities.

• **Dynamicity** - Minimize the complexity of using the dynamicity of services

• **Business Logic** - A focus on writing business logic in regular Java classes that are not required to implement certain framework APIs or contracts in order to integrate with a container.

• **Declarative** - This facilitates independent testing of components and reduces environment dependencies.

• **Familiarity** - Familiar to enterprise Java developers.

### 121.1.2 Entities

• **Blueprint Extender** - The bundle that creates and injects component instances for a Blueprint bundle as configured in that Blueprint bundle’s XML definition resources.

• **Blueprint Container** - Represents the activities of the Blueprint Extender for a specific Blueprint Bundle.

• **Blueprint Bundle** - A bundle that is being constructed by the Blueprint Container because it has a Bundle-Blueprint header or it contains XML resources in the OSGI-INF/blueprint directory.

• **Manager** - A manager is responsible for the life cycle of all component instances for one component definition. There are the following types of managers, A manager is a bean manager, a service reference manager, or a service manager. A manager can have explicit and implicit dependencies on other manager. During instantiation and runtime, a manager can provide a component instance to be injected or used in other ways.

• **Component** - A loosely defined term for the application building blocks and their infrastructure. Components are instantiated into component instances by a manager that is configured with a Component Metadata subclass that is derived from a Component Definition.

• **Component Instance** - An object that is part of the application. Component Instances are created and managed by their component manager.

• **Component Definition** - Configuration data used by a manager to construct and manage component instances. This configuration data is represented in Metadata, an interface hierarchy starting with the Metadata interface.

• **Bean Manager** - A manager that has metadata for creating Java objects and injecting them with objects and component instances that come from other managers it implicitly depends on.

• **Service Manager** - A manager that handles the registration of a service object that is provided by a component instance.

• **Service Reference Manager** - The general name for the reference and reference-list managers.

• **Reference Manager** - A manager that handles the dependency on a single OSGi service.

• **Reference-list Manager** - A manager that handles the dependency on a list of OSGi services.

• **Environment Manager** - A manager that can provide information from the Bundle’s environment. For example, the BlueprintContainer object is made available through an environment manager.

• **Target** - A manager type useful in a callback context. These are the ref (which is an indirection to), a reference, and a bean manager.

• **Property** - A conceptual instance variable of a component instance provided by a bean manager that is set on the component instance with a corresponding set<Name> method.

• **Argument** - Metadata for an argument in a constructor or method.
• Type Converter - A component instance defined, or referenced, in the type-converters section implementing the Converter interface.

Figure 121.1  Blueprint Class and Service Overview

121.1.3 Synopsis

The Blueprint Extender bundle waits for Blueprint bundles. These are bundles that contain Blueprint XML resources called the definitions. These XML resources can be found in a fixed location or pointed to from a manifest header. When a Blueprint extender bundle detects that a Blueprint bundle is ready, it creates a Blueprint Container to manage that Blueprint bundle.

The Blueprint Container then parses the definitions into metadata objects. All top-level elements in the definitions are ComponentMetadata objects and are registered in the Blueprint Container by their id.

For each of the ComponentMetadata objects, the Blueprint Container has a corresponding component manager. For example, a BeanMetadata object relates to a Bean Manager instance. There are the following types of managers:

- Bean Managers - Can provide general objects that are properly constructed and configured
- Service Managers - Can register services
- Service Reference Managers - Provide proxies to one or more services. There are two sub-types: reference-list and reference.
- Environment Managers - Holding environment values like the Blueprint Bundle object

After creation, all managers are not yet activated. A manager is activated on demand when it has to provide a component instance for the first time.

All service reference managers track services in the service registry in order to determine if they are satisfied or not. If not, the Blueprint Container can optionally start a grace period. During the grace period, the Blueprint Container waits for all mandatory service reference managers to become satisfied. If this does not happen during the grace period, the Blueprint Container must abort the initialization.

From now on, the Blueprint Container is ready to provide component instances. Whenever a manager is asked to provide a component instance for the first time, the manager is activated. This acti-
vation will first request all its dependencies to provide a component instance, activating these managers if not already activated, recursively.

However, the activation needs a trigger to start. There are two triggers.

- **Service Request** - All service managers must have a Service Factory registered with the OSGi service registry whenever that service manager is enabled, see *Enabled* on page 272.
- **Eager Managers** - To kick start the application in the bundle, the Blueprint Container must ask all eager managers to provide a component instance, thereby activating these managers, see *Eager Instantiation* on page 254.

Service references must actuate their reference listeners when they are activated.

Bean managers have a scope. This scope can be singleton, where the manager always provides the same object, or prototype, where the manager creates a new object for each request.

Service reference managers provide proxies to the actual service objects and fetch the service object lazily. They provide a constant reference that dampen the dynamics of the underlying service objects.

If the Blueprint Container has successfully activated the eager managers, it will register a Blueprint Container service.

When the Blueprint Container must be destroyed because: the Blueprint bundle has stopped, there is a failure, or the Blueprint extender is stopped, then the Blueprint Container service is unregistered and all managers are deactivated. This will unregister any services and disable listeners, which release the component instances. Then all component instances are destroyed in reverse dependency order. That is, a component instance is destroyed when no other component instances depend on it.

### 121.2 Managers

The key feature of the Blueprint Container specification is to let the application in the bundle be constructed in the proper order from objects that are not required to be aware of Blueprint, OSGi, or even each other. These objects are called **component instances**. The active entity that orchestrates the life cycle of the bundle application is the **Blueprint Container**. It is configured by XML resources in the Blueprint bundle. The Blueprint Container is responsible for construction and configuration of the component instances as well as the interaction with the service registry.

Inside the Blueprint Container, component instances are managed by a **manager**. A manager is configured with one Component Definition, for example a bean definition, and can then provide one or more component instances. Such a configured manager instance is also loosely called a **component**.

A manager can have additional behavior associated with it. This behavior is controlled by the manager's **type**. This specification defines a number of manager types: bean, service, environment, reference, and reference-list. These types are further defined in the next section.

These managers are conceptual, they are not visible in the API of this specification. That is, an implementation is free to implement the specification without these objects as long as the externally observable behavior is the same.

As an example, a trivial echo service:

```xml
<blueprint>
  <service id="echoService"
    interface="com.acme.Echo" ref="echo"/>
  <bean id="echo" class="com.acme.EchoImpl">
    <property name="message" value="Echo: "/>
  </bean>
</blueprint>
```
public interface Echo {
    public String echo(String m);
}

class EchoImpl implements Echo {
    String message;
    public void setMessage(String m) {
        this.message = m;
    }
    public void echo(String s) { return message + s; }
}

The example defines two top-level managers: echoService and echo. The echoService manager is of type service, and the echo manager is of type bean. The service manager is responsible for registering an OSGi service, where the service object will be the component instance provided by the echo manager. The echo component instance gets a message injected.

As seen from the example, managers can use component instances from other managers to construct their component instances. The use of other managers creates an implicit dependency. Managers can also declare explicit dependencies. Dependencies are transitive, see Manager Dependencies on page 245 for more information. In the previous example, the echoService service manager depends on the echo manager, this is an implicit dependency.

Managers have their own life cycle. They are conceptually created after the Blueprint Container has decided to run the application, see Blueprint Life-Cycle on page 249. However, the intention of this specification is to allow the bundle application to lazily activate. That is, no application code is used until there is a trigger like a service request or a service manager has an explicit dependency. A manager must always be atomically activated before it provides its first component instance. During activation, listeners are actuated and notified, service objects are requested, etc. The details are described in the appropriate manager's type description.

Each manager type has an associated component metadata type. Component Metadata is used to configure a manager. XML definition resources in the bundle define the source for this Metadata. In the previous example, the service and bean XML element are translated to a ServiceMetadata and BeanMetadata object respectively.

The Blueprint Container maintains a registry of managers by their id. These are the managers that are called the top-level managers. Top level managers are managers defined as child elements of the top XML blueprint element or bean managers in the type-converters element. Their Metadata is registered under their id (or calculated id) in the Blueprint Container. All top level managers share a single namespace. That is, it is an error if the same id is used multiple times or attempts to override the built-in environment managers.

Top level managers can depend on other top level managers but there are many places where a manager can depend on an inlined manager. In these places, a complete manager can be defined inside another manager. Such inlined managers are always anonymous: they must not have an id and must not be registered as a top-level manager. Inlined beans are further constrained to always have prototype scope. That is, every time they are asked to provide a component instance, they must return a different object.

When the Blueprint Container must be destroyed, all singleton component instances that have been created must be destroyed. This must first deactivate all activated managers. All these managers must release their dependencies on any component instances they hold. Then the Blueprint Container must destroy all singleton component instances. The order of this destruction must be such that a component instance is only destroyed if there are no other component instances depending on it. See Reverse Dependency Order on page 246.
The relations between manager types, component instances, metadata and the Blueprint Container is schematically depicted in Figure 121.2 on page 244.

**Figure 121.2 Managers and Metadata**

![Diagram showing the relations between manager types, component instances, metadata and the Blueprint Container](image)

### 121.2.1 Manager Types

Blueprint only supports a fixed set of the following manager types:

- **Bean** - A bean manager provides regular Java objects as component instances. It has the following features:
  - Construction via class name, static factory method, or a factory method on a target. A target is a reference to a top level manager of type bean or service reference, or a referral to a top level manager of those types.
  - Can have arguments for a constructor or factory method.
  - Can have properties that are injected.
  - Manages a singleton or creates objects on demand depending on its scope.
  - Life cycle callbacks for end of initialization and destruction.

  See [Bean Manager](page 260) for more details.

- **Reference** - Reference managers track a service in the OSGi service registry. When activated, they provide a proxy to a service object. See [Service Reference Managers](page 272) for more details. A reference is satisfied when its selection matches a service in the registry.

- **Reference-list** - Reference-list managers track multiple services. A reference-list is satisfied when its selection matches one or more services in the registry. See [Service Reference Managers](page 272) for more details.

- **Service** - Service managers maintain the registration of an OSGi service object. Service managers provide a proxied ServiceRegistration object so that the application code has a constant reference, even if the service is unregistered and then registered again. A service manager is enabled
if all the mandatory service references in its dependencies are satisfied. See Service Manager on page 266.

- **Environment** - Environment managers provide access to the environment of the Blueprint bundle, for example its Bundle Context. See Blueprint Container on page 294 for more details.

### 121.2.2 Metadata Objects

Metadata objects hold the configuration information (from the Component Definition) for the managers. These metadata objects represent the element structure found in the XML definitions in canonical form. Each element in the XML has a corresponding Metadata sub-type that has a name that maps directly to the element. For example, the bean element represents the bean manager that has its configuration data defined in the BeanMetadata interface.

There are Metadata interfaces for all the manager types, except the environment type. Some dependency injections require the construction of arrays, maps, properties, simple objects, etc. For these type of objects, additional Metadata sub-interfaces are defined; these interfaces provide the information to construct the basic programming types. For example, the CollectionMetadata interface contains the information to construct an Array or Collection of a given type, where its member values are defined by other Metadata objects.

The set of Metadata types is fixed in this specification, just like the set of manager types. It is impossible to extend this set with user defined Metadata types. For more information about Metadata, see Metadata on page 299.

### 121.2.3 Activation and Deactivation

Managers are created after all the definitions are parsed. Some managers can already show some activity, for example service managers always activate explicit dependencies and register a Service Factory with the OSGi service registry. However, in this state a manager should attempt to not use any resources from the Blueprint bundle until it is activated itself.

A manager must be atomically activated when it has to provide its first component instance. During activation it can perform a manager specific initialization that will actually consume resources from the Blueprint bundle. This activation must be atomic. That is, if a manager is being activated then other threads must block until the activation is completed.

Deactivation only happens during the destruction of the Blueprint Container. During deactivation, a manager must release any dependencies on resources of the Blueprint bundle. No components instances are destroyed during deactivation because the singleton component instance destruction must happen after all managers are deactivated.

Each manager type has a dedicated section that describes what must happen during its activation and deactivation.

### 121.2.4 Manager Dependencies

Managers that refer to other managers depend on these managers transitively. For example, a service manager depends directly on the manager that provides the service object. In its turn, that service object could depend on any provided objects that were used to construct and inject this service object, and so on. This transitive set of dependencies are called implicit dependencies because these dependencies are implicitly created by the use of other managers in the Component Definitions.

Managers can also be configured with explicit dependencies. The XML definitions for all managers have a depends-on attribute with a whitespace delimited list of manager ids. Each of these depends-on managers must provide an object, that will be ignored. The timing of activation of dependencies depends on the specific managers but in general should happen before any observable behavior.

There is no ordering guarantee between independent sets of dependencies. The dependency graph is based on the managers, not the component instances. For example, the following definition:
After initialization, there will be the following component instances: \(a\), \(b\), \(d\), \(e\), and three \(c\)’s. Lower case names are used for instances, the corresponding upper case is its manager. The ordering guarantee is that manager \(D\) is activated before manager \(C\), manager \(C\) is activated before manager \(E\) and \(B\), manager \(E\) is activated before manager \(B\), and manager \(B\) is activated before manager \(A\). There will be no component instance \(f\) created because \(F\) is a lazy manager. There are three \(c\)’s because manager \(E\) and \(B\) have an implicit dependency on \(C\) and manager \(B\) has an additional explicit dependency, totaling 3 dependencies. One of these \(c\)’s is an orphan and will be garbage collected over time because it is not referred to by any component instance.

The example is depicted in Figure 121.3 on page 246.

**Figure 121.3 Dependency Graph after initialization**

**121.2.5 Reverse Dependency Order**

The destruction of component instances must be done in **reverse dependency order**. This concept is defined as only destroying a singleton component instance (in a manager specific way) when no other activated singleton component instance has an implicit or explicit dependency on it. That is, a component instance has no more field references to other component instances. A component that never was activated does not have any dependencies.

This strategy will ensure that a component instance cannot have an instance field that refers to a component instance that has been destroyed.

Deactivating the manager will release its dependencies, which then frees up other component instances until all component instances are destroyed, or there are cyclic references. In the case of cyclic dependencies, the order of destruction is undefined.

In the example depicted in Figure 121.3 on page 246, the previous rules imply that component instance \(a\) can be immediately destroyed first because it has no clients. After component instance \(a\) is destroyed, component instance \(b\) becomes free because no other component instances refer to it. The explicit dependency from manager \(F\) to manager \(B\) was never activated, so it is not taken in-
to account. The destruction of component instance b frees up component instance e and c because now the explicit dependency from manager B to manager E and manager B to manager C have been released. Manager C is deactivated but no component instances are destructed because it has prototype scope; these managers do not destroy their component instances. Then component instance d can be destructed.

121.2.6 Cyclic Dependencies

The implicit and explicit dependencies of a component form a dependency graph. In the ideal case, this graph should be free from cycles. A cycle occurs when a set of one or more managers find themselves in their own implicit or explicit dependencies. For example:

```java
class A { public A(B b); }
class B { public void setA(A a); }

<bean id="a" class="A"> <argument ref="b"/> </bean>
<bean id="b" class="B"> <property name="a" ref="a"/> </bean>
```

In this example, the cycle is the set {a,b}. Managers can be part of multiple cycles.

When a member of a cycle is requested to provide a component instance, the Blueprint Container must break the cycle by finding one breaking member in the cycle's members. A breaking member must be a singleton bean and use property injection for the dependency that causes the cycle. The Blueprint Container can pick any suitable member of the cycle for breaking member, if no such member can be found, then initialization fails or the getComponentInstance method must throw a Component Definition Exception.

In the previous example, manager b can be a breaking member because it uses the property injection for the cyclic dependency on manager a. Manager a cannot be a breaking member because the cyclic dependency is caused by a constructor argument, a breaking member must use property injection for the cyclic dependency to be broken.

A breaking member must return a partially initialized component instance when it is asked to provide an object. A partially initialized object has done all possible initialization but has not yet been called with the initMethod (if specified) nor has it been injected any of the properties that causes a cycle. The finalization of the partially initialized component instance must be delayed until the breaking member has been injected in all referring members of the cycles. Finalization means injecting any remaining unset properties and calling of the initMethod, if specified.

The consequence of partially initialized component instances is that they can be used before they have all properties set, applications must be aware of this.

All partially initialized component instances must be finalized before the Blueprint Container enters the Runtime phase and before a call to the getComponentInstance method returns a component instance.

All detected cycles should be logged.

Consider the following example:

```java
class A {
    public A(B b) {}
}
class B {
    public B(A a) {}
}
```

And the configuration:

```xml
<bean id="a" class="A"> <argument ref="b"/> </bean>
```
In this case, the cycle cannot be broken because neither manager qualifies as breaking manager because they have a constructor/factory argument dependency. That is, it is impossible to construct an object without using the dependency. However, consider the following example:

```java
public class A {
    public A(B b) {}
}
public class B {
    public B(C c) {}
}
public class C {
    public void setA(A a) {}
}
```

And the configuration:

```xml
<bean id="a" class="A"> <argument ref="b"/> </bean>
<bean id="b" class="B"> <argument ref="c"/> </bean>
<bean id="c" class="C" init-method="done">
    <property name="a" ref="a"/>
</bean>
```

This configuration is depicted in Figure 121.4 on page 248. This cycle \{a,b,c\} can be broken by selecting manager c as the breaking member. If manager a is requested to provide a component instance for the first time, then the following sequence takes place:

```java
activate a
activate b
activate c
    c = new C()
    b = new B(c)
    a = new A(b)
    c.setA(a)
    c.done()
return a
```

Cycles must be broken, if possible, both for singleton managers as well as prototype beans, although a breaking manager must always be a singleton bean because a prototype bean must always return a new object, making it impossible to break the cycle by returning a partially initialized component instance. That is, the following definition is not allowed to attempt to create an infinite loop:

```xml
<bean id="a" scope="singleton" class="A">
    <property name="a" ref="a"/>
</bean>
```

The previous definition must create an A object that refers to itself. However, if the example had used a prototype scope, it would be an unbreakable cycle.
121.2.7 Eager Managers

The Blueprint Container can force the activation of the application in the Blueprint bundle with eager managers. An eager manager is a manager that has the activation set to eager. A bean manager can only be eager if it has singleton scope.

Eager managers are explicitly activated by asking them to provide a component instance after all other initialization is done. A bundle that wants to be lazily initialized should not define any eager managers.

121.3 Blueprint Life-Cycle

A bundle is a Blueprint bundle if it contains one or more blueprint XML definition resources in the OSGI-INF/blueprint directory or it contains the Bundle-Blueprint manifest header referring to existing resources.

A Blueprint extender is an implementation of this specification and must track blueprint bundles that are type compatible for the Blueprint packages and initialize them appropriately. The timing and ordering of the initialization process is detailed in the following section.

There should be only one Blueprint extender present in an OSGi framework because this specification does not specify a way to resolve the conflicts that arise when two Blueprint extenders extend the same Blueprint bundle.

121.3.1 Class Space Compatibility

A Blueprint extender must not manage a Blueprint bundle if there is a class space incompatibility for the org.osgi.service.blueprint packages. For example, if the Blueprint bundle uses the Blueprint-Container class, then it must import the org.osgi.service.blueprint.container package. The Blueprint extender and the Blueprint bundle must then share the same class space for this package. Type compatibility can be verified by loading a class from the blueprint packages via the Blueprint extender bundle and the Blueprint bundle’s loadClass methods. If the Blueprint bundle cannot load the class or the class is identical to the class loaded from the extender, then the two bundles are compatible for the given package. If the Blueprint extender is not class space compatible with the Blueprint bundle, then Blueprint extender must not start to manage the Blueprint bundle.

121.3.2 Initialization of a Blueprint Container

A Blueprint extender manages the application life cycle of Blueprint bundles based on:

- The Blueprint bundle state,
- The Blueprint definitions,
- The Blueprint extender’s bundle state
- The class space compatibility

All activities on behalf of the Blueprint bundle must use the Bundle Context of the Blueprint bundle. All dynamic class loads must use the Blueprint bundle’s Bundle loadClass method.

The following sections describe a linear process that handles one Blueprint bundle as if it was managed by a special thread, that is, waits are specified if the thread waits. Implementations are likely to use a state machine instead for each managed Blueprint bundle, the linear description is only used for simplicity.

In the following description of the initialization steps, the Blueprint Container will update its state. State changes are broadcast as events, see Events on page 295.
If any failure occurs during initialization, or the Blueprint bundle or Blueprint extender bundle is stopped, the Blueprint Container must be destroyed, see Failure on page 250. These checks are not indicated in the normal flow for clarity.

121.3.2.1 Initialization Steps

The initialization process of a Blueprint Container is defined in the following steps:

1. Wait until a blueprint bundle is ready. A blueprint bundle is ready when it is in the ACTIVE state, and for blueprint bundles that have a lazy activation policy, also in the STARTING state.
2. Prepare, verify if this Blueprint bundle must be managed, see Preparing on page 252.
3. State = CREATING
4. Parse the XML definition resources.
5. Service reference managers must start tracking their satisfiability without actually activating. See Tracking References on page 253.
6. If all mandatory service references are satisfied, or the blueprint.graceperiod is false, then go to step 9.
7. State = GRACE_PERIOD
8. Perform the grace period. This period waits until all mandatory service references are satisfied. See Grace Period on page 253. This step fails if the mandatory dependencies are not satisfied at the end of the grace period.
9. The Blueprint Container is now ready to provide component instances.
10. Service managers must initialize their explicit dependencies and have a Service Factory registered during the periods that they are enabled. See Service Registration on page 253.
11. Ask all eager managers to provide a component instance. See Eager Instantiation on page 254.
12. State = CREATED
13. Register the Blueprint Container
14. The components are now active and perform their function until the Blueprint bundle or the Blueprint extender bundle are stopped.
15. State = DESTROYING
16. Perform the Destroy phase, see Destroy the Blueprint Container on page 254.
17. State = DESTROYED

121.3.2.2 Failure

If at any time there is a failure, the Blueprint Container must:

1. State = FAILURE
2. Unregister the Blueprint Container service.
3. Destroy the Blueprint Container.
4. Wait for the Blueprint bundle to be stopped.

121.3.2.3 Diagram

This initialization process is depicted in Figure 121.5 on page 251.
121.3.3 Extensions

A compliant implementation of this specification must follow the rules as outlined. However, implementations can provide functional extensions by including attributes or elements of other namespaces. For example, a Blueprint extender implementation that supports proxying of certain classes and a number of additional type converters could include a \texttt{http://www.acme.com/extensions} namespace that adds an \texttt{extensions} attribute on the blueprint element:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<blueprint
 xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
 xmlns:ext="http://www.acme.com/extensions"

 ext:extensions="proxyClasses"
>
...
</blueprint>
```
Blueprint extenders that detect the use of an unrecognized namespace must fail to signal a portability problem.

### 121.3.4 Preparing

Blueprint definitions are stored as resources in the Blueprint bundle. If a Bundle-Blueprint manifest header is defined, then this header contains a list of paths. The Bundle-Blueprint header has the following syntax:

```
Bundle-Blueprint ::= header
    // See Common Header Syntax in Core
```

This specification does not define any attributes or directives for this header. Implementations can provide proprietary parameters that should be registered with the OSGi Alliance to prevent name collisions. The non-localized version of the header must be used.

The last component of each path in the Bundle-Blueprint header may use wildcards so that `Bundle.findEntries` can be used to locate the XML document within the bundle and its fragments. The `findEntries` method must always be used in the non-recursive mode. Valid paths in the header have one of the following forms:

- **absolute path** - The path to a resource in the fragment or directory, this resource must exist. For example `cnf/start.xml`.
- **directory** - The path to directory in a fragment or main bundle, the path must end in a solidus (`/`). The pattern used in the `findEntries` method must then be `*.xml`. The directory is allowed to be empty.
- **pattern** - The last component of the path specifies a filename with optional wildcards. The part before is the path of directory in the bundle or one of its fragments. These two parts specify the parameter to `findEntries`. It is allowed to have no matching resources. An example of a pattern is: `cnf/*.xml`.

If no resources can be found, then the Blueprint bundle will not be managed and the initialization exits.

For example, the following header will read the resources `/lib/account.xml`, `/security.bp`, and all resources which path ends in `.xml` in the `/cnf` directory:

```
Bundle-Blueprint: lib/account.xml, security.bp, cnf/.*.xml
```

If the Bundle-Blueprint header is not defined, then its default value is:

```
OSGI-INF/blueprint/.*.xml
```

A Bundle-Blueprint manifest header specified in a fragment is ignored by the Blueprint Container. However, XML documents referenced by a bundle's Bundle-Blueprint manifest header, or its default, may be contained in attached fragments, as defined by the `findEntries` method.

If the Bundle-Blueprint header is specified but empty, then the Blueprint bundle must not be managed. This can be used to temporarily disable a Blueprint bundle.

### 121.3.5 Parsing

The Blueprint Container must parse the XML definitions into the Blueprint Container's metadata registry. Parsing fails if:

- A path from the Bundle-Blueprint header cannot be found in the bundle or any of its fragments.
- An XML definition does not validate against its schema.
- The XML elements do not meet one or more of their constraints
- Any errors occur
121.3.6 Tracking References

Service reference managers must track the service registry to see if they are satisfied or not. These managers must not be activated to register these service listeners nor must they activate any dependencies until they are activated. That is, no component instances for the reference listeners are obtained until the service reference manager is activated.

121.3.7 Grace Period

A Blueprint Container by default will wait for its dependencies in the grace period. However, this can be overridden with a directive on the Bundle-SymbolicName header of the Blueprint bundle:

- `blueprint.graceperiod (true|false)` - If set to `true`, then the Blueprint Container must enter the grace period and wait for dependencies, this is the default. Otherwise, it must skip the grace period and progress to the next phase regardless if there are any unsatisfied service references.

The purpose of the grace period is to handle the initialization of multiple bundles gracefully. The grace period will first wait a configurable time for all mandatory service references to become satisfied, or for the bundle to stop. If these mandatory services are satisfied, then the grace period succeeds, otherwise it will fail. If the bundle is stopped during the grace period, then the Blueprint Container must be destroyed.

During the waiting period services can come and go. Each time such a service event takes place that involves any of the mandatory service references, the Blueprint Container must send out another GRACE_PERIOD event if that event does not result in ending the grace period. The event contains the complete filters of the unsatisfied service references, see Blueprint Event on page 296.

The wait time for the grace period is defined in a directive on the Bundle-SymbolicName header of the Blueprint bundle:

- `blueprint.timeout (Integer >= 0)` - The time to wait in the grace period for dependencies to become satisfied in milliseconds. The default is 300000, which is 5 minutes. If the timeout is 0, an indefinite wait will take place.

OSGi services are dynamic, therefore the grace period does not guarantee that all mandatory service references are still available. It only guarantees that at one moment in time they were available. A mandatory reference can become unsatisfied at any moment in time when a service is not available. See the Service Dynamics on page 292 for a description of how this is handled.

For example, the following header will make the bundle wait a maximum of 10 seconds for its mandatory service references to be satisfied. These dependencies must be satisfied, or a failure occurs.

```
Bundle-SymbolicName: com.acme.foo;
  blueprint.graceperiod:=true;
  blueprint.timeout:= 10000
```

121.3.8 Service Registration

A service manager must first activate all its explicit dependencies but it must not activate. It must then ensure that a Service Factory object is registered as a service when that service is enabled. Enabled means that all of the mandatory service references in its dependencies are satisfied.

Once the Service Factory is registered, any bundle can get the corresponding service object. Such a request must activate the service manager, if it is not already activated. Activation of a service manager must obtain a component instance from the Blueprint Container for the service object and any registration listeners. The registration listeners are then actuated and notified of the initial state.
121.3.9 Eager Instantiation

After all initialization is done, the Blueprint Container is ready. It is now possible to request component instances. If a bundle needs immediate startup because they cannot wait until they are triggered, then it should set the activation of its bean managers to eager. The Blueprint Container must request all eager managers to provide a component instance in this instantiation phase, see also Lazy and Eager on page 259.

121.3.10 Runtime Phase

The Blueprint Container must be registered as a service with the following service properties:

- osgi.blueprint.container.symbolicname - The bundle symbolic name of the Blueprint bundle
- osgi.blueprint.container.version - The version of the Blueprint bundle

The Blueprint Container service must only be available during the runtime phase when initialization has succeeded.

As long as the Blueprint extender and the Blueprint bundle are active, the application is in the runtime phase. The component instances perform their requested functionality in collaboration. The Blueprint Container can be used to provide objects from the defined managers, get information about the configuration, and general state information, see Blueprint Container on page 294.

121.3.11 Destroy the Blueprint Container

The Blueprint Container must be destroyed when any of the following conditions becomes true:

- The Blueprint bundle is stopped, that is, it is no longer ready.
- The Blueprint extender is stopped
- One of the initialization phases failed.

Destroying the Blueprint Container must occur synchronously with the Bundle STOPPING event if that caused any of the previous conditions. For example, if the Blueprint extender is stopped, it must synchronously destroy all Blueprint Containers it has created.

Destroying the Blueprint Container means:

1. Unregistering the Blueprint Container service
2. Deactivating all managers.
3. Destroying all component instances in reverse dependency order, see Reverse Dependency Order on page 246.

A Blueprint Container must continue to follow the destruction even when component instances throw exceptions or other problems occur. These errors should be logged.

If the Blueprint extender is stopped, then all its active Blueprint Containers must be destroyed in an orderly fashion, synchronously with the stopping of the Blueprint extender bundle. Blueprint Containers must use the following algorithm to destroy multiple Blueprint Containers:

1. Destroy Blueprint Containers that do not have any services registered that are in use by other bundles. More recently installed bundles must be destroyed before later installed bundles, that is, reverse bundle id order.
2. The previous step can have released services, therefore, repeat step 1 until no more Blueprint Containers can be destroyed.
3. If there are still Blueprint Containers that are not destroyed, then destroy the Blueprint Container with:
   - The registered service that is in use with the lowest ranking number, or if a tie
   - The highest registered service id
If there are still Bundle Containers to be destroyed, retry step 1

During the shutting down of an OSGi framework, it is likely that many bundles are stopped near simultaneously. The Blueprint extender should be able to handle this case, without deadlock, when the stop of a Blueprint bundle overlaps with the stop of the Blueprint extender bundle.

### 121.3.12 Failure

If a failure occurs during the initialization of the Blueprint bundle, then first a FAILURE event must be posted, see Events on page 295. Then the Blueprint Container should be destroyed, ensuring that no uninitialized or half initialized objects are destroyed. Failures should be logged if a Log Service is present.

### 121.3.13 Lazy

The Blueprint Container specification specifically allows lazy initialization of the application in the Blueprint bundle. No component instances are created until an eager manager is activated, or a service request comes in.

If no eager managers are defined and no service has explicit dependencies, then no component instances are provided until an external trigger occurs. This trigger can be a service request or a call to the getComponentInstance method of the Blueprint Container, which is registered as a service. This allows a Blueprint bundle to not create component instances, and thereby load classes, until they are really needed. This can significantly reduce startup time.

Some features of the component definitions can only be verified by inspecting a class. This class loading can break the lazy initialization of a Blueprint bundle. It is therefore allowed to delay this kind of verification until the activation of a manager.

This lazy behavior is independent of the bundle's lazy activation policy. Though the Blueprint extender recognizes this policy to detect when the bundle is ready (for a lazy activated bundle the STARTING state is like the ACTIVE state), it is further ignored. That is, the relation between a Bundle Activator that is lazily activated and the Blueprint Container is not defined.

### 121.4 Blueprint Definitions

The Blueprint XML resources in a bundle are the definitions. Each definition can include multiple namespaces. Implementations of the Blueprint core namespace must strictly follow this specification, if they add additional behavior they must add additional namespaces that are actually used in the definitions to signal the deviation from this specification.

The namespace for the core Blueprint definition resources is:

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

Blueprint resources that use this core specification must have as top the blueprint element. The following example shows the body of a Blueprint definition:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<blueprint

 xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

 ...
</blueprint>
```

The recommended prefix for the Blueprint core namespace is bp.

All elements in the Blueprint namespace are prepared for future extensions and provide a description child element in most positions.
121.4.1 XML

In the following sections, the XML is explained using the normal syntax notation used for headers. There is, however, one addition to the normal usage specific to XML, and that is the use of the angled brackets (<>). A term enclosed in angled brackets, indicates the use of a real element. Without the angled brackets it is the definition of a term that is expanded later to a one or more other terms or elements. For example:

```
people     ::= <person> *
person     ::= <child>* address
address    ::= <fr> | <us> | <nl>
```

Describes for example the following XML:

```
<people>
  <person id="mieke">
    <child name="mischa"/>
    <child name="thomas"/>
    <fr zip="34160"/>
  </person>
</people>
```

Attributes are described in tables that define how they map to their corresponding Metadata. As a rule, the XML elements and attributes are expressed directly in the Metadata.

The text in the following sections is a normative description of the semantics of the schema. However, the structure information is illustrative. For example, all description elements have been ignored for brevity. The exact structure is described by the XML schema, see Blueprint XML Schema on page 300.

There are a number of convenient XML types used in the following sections. There schema types are defined here:

- `qname` - A fully qualified Java class name in dotted form, for example `java.lang.String`.
- `method` - A valid Java method name, for example `setFoo`.
- `type` - A name of a Java type including arrays, see the next section Syntax for Java types on page 256.
- `target` - An inline bean, reference, or ref, see Target on page 259.
- `object` - An object value, see Object Values on page 279.

In several cases, the actual syntax depends on the type conversion. This type of syntax is indicated with `<type>` indicates that the syntax of the string depends on the type conversion, where ten type is usually given as a parameter on the same Metadata.

121.4.2 Syntax for Java types

A number of elements can refer to a Java type, for example the value element has a type attribute and a map element has a key-type attribute. The syntax for these types is as follows:

```
type    ::= qname array
array    ::= '[]' *
```

Where `qname` is the fully qualified name of a Java class or interface, or the name of a primitive type.

For example:

```
<value type="java.lang.String[]"/>
```
It is not possible to specify generic information in this syntax.

### 121.4.3 XML and Metadata

The Blueprint Container parses the XML into Metadata objects, see Metadata on page 299. During parsing, the XML parser validates against the detailed Blueprint schema and will therefore catch many errors. However, the XML schema and the Metadata type are not equivalent. The XML contains many conveniences that the Blueprint Container must convert to the canonical type in the Metadata. A number of general rules apply for this conversion:

- An absent attribute will result in null, unless the schema element provides a default value. In that case, the default must be returned from the Metadata object. That is, a default is indistinguishable from a specifically set value.
- Defaults from the blueprint element are filled in the Metadata objects, they are not available in any other way.
- Strings are trimmed from extraneous whitespace, as described in XML normalization.
- Child elements are represented by List objects, in the order of their definition. If no child elements are specified, the list will be empty.

For example, the activation feature reflects the total of default-activation and activation attributes but does not reflect that a prototype scope always makes a bean lazy. That is, even if activation is eager, the bean must still have lazy activation when it has prototype scope.

### 121.4.4 `<blueprint>`

The blueprint element is the top element. The definitions consist of two sections: the type-converters section and the managers section.

```
blueprint ::= <type-converters>manager*
manager ::= <bean> | <service>
           | service-reference
service-reference ::= <reference> | <reference-list>
type-converters ::= <bean> | <ref>
```

In this specification, the reference and reference-list managers are referred to as service references when their differences are irrelevant. The blueprint element structure is visualized in Figure 121.6.

![Diagram of Blueprint Element Structure](image)

**Figure 121.6** Managers (bold = element name, plain=base type)

= choice

### 121.4.5 Metadata

The blueprint element has no corresponding Metadata class.
121.4.6 Defaults

The blueprint element supports the setting of the diverse defaults for the current definition resource with the following attributes:

- **default-activation** - Controls the default for the activation attribute on a manager. See *Lazy and Eager* on page 259. The default for this attribute is eager.
- **default-availability** - The default availability of the service reference elements, see *Service Reference Managers* on page 272. The default for this attribute is mandatory.
- **default-timeout** - The default for the reference element timeout attribute, see *Service Reference Managers* on page 272. The default for this attribute is 30000, or 5 minutes.

These defaults are specific for one definition resource, they apply only to elements enclosed to any depth in the blueprint element. These defaults are not visible in the Metadata.

121.4.7 `<type-converters>`

The Blueprint definitions are text based but the component instances require actual classes for their construction and dependency injection. Component instances are injected with general objects the target type is not always compatible with the source type. This specification therefore allows for type conversion. Type conversion rules are specified in *Type Conversion* on page 287. This section provides beans, or referrals to beans, that can be used in this type conversion process. They are listed in a separate section so they can be registered as a type converter, pre-instantiated, and preventing dependencies that easily become cyclic. Beans defined in the `<type-converters>` element must be registered as top-level managers.

The structure of the `<type-converters>` element is:

```xml
<type-converters> ::= ( <bean> | <ref> )*
```

Type converters defined with the `<ref>` element can refer to bean managers or reference managers. Type converters must have ids distinct from any other manager and are available through the Blueprint Container’s `getComponentInstance` method.

121.4.8 manager

The component XML schema type is the base type of the bean, service, reference-list, and reference elements. All manager sub-types share the following attributes:

- **id** - The manager and its Metadata are identified by its id as defined in its Component Definition. In general this id is therefore referred to as the *component id*. This is an optional attribute. If it is not defined, a default calculated unique id will be assigned to it for top-level managers. For inlined managers, the id attribute cannot be set, their Metadata must return null. All top level manager ids must be unique in a Blueprint Container.

  The id attribute must be of type ID as defined in XML Schema, see [8] *XML Schema*. The syntax for an id is therefore:

  ```xml
  id ::= ID // See ID in [8] XML Schema
  ```

  Ids generally use camel case, like `myComponent`, and they are case sensitive. That is, component id `madHatter` and `madhatter` are distinct ids. Applications should not use ids starting with the prefix `blueprint`.

  Ids are not required, if no component id is specified, the Blueprint Container must assign a unique id when it is a configured in a top level element. This calculated id must start with a full stop (".").

- **activation** - Defines the activation mode to be lazy or eager. See *Eager Instantiation* on page 254.
• dependsOn - The list of explicit dependencies that must be activated. See Explicit Dependencies on page 259.

The Metadata interface of top level managers will be a sub-interface of ComponentMetadata and is available from the Blueprint Container by its component id.

Figure 121.7 Inheritance hierarchy for managers

121.4.9 Explicit Dependencies

The dependsOn list contains the ids of the top-level managers the bean explicitly depends on. Unless stated otherwise in the specific manager description, explicit dependencies must be activated before their manager is activated.

For example:

```xml
<bean id="alice" class="com.acme.MadHatter"
     depends-on="cheshire rabbit queen"/>
```

This example will ask the top level managers cheshire, rabbit, and queen to provide an object before alice is activated. For a discussion about dependencies see Manager Dependencies on page 245.

121.4.10 Lazy and Eager

During initialization, all eager top level managers are requested to provide a component instance. Applications can use this request as an indication to start providing their intended functionality.

Managers that are lazy, that is, not singleton scope, activation is lazy, or inlined, are activated when they are first asked to provide a component instance. Therefore, even lazy managers can activate during initialization when they happen to be a dependency of another manager that activates its dependencies.

Services and service references can also have lazy or eager activation. The eager activation will ensure that all listeners are properly actuated during the corresponding activation. For services, the service object is then also requested at startup.

The following example defines an eager bean by making it a singleton and setting the activation to eager:

```xml
<bean id="eager" scope="singleton"
     class="com.acme.FooImpl" activation="eager"/>
```

121.4.11 Target

In several places in the Blueprint schema it is necessary to refer to a target. A target is a:
Bean Manager

A bean manager provides an arbitrary Java object. It constructs this object from a given class or factory and then configures the object by injecting its properties with other component instances or more general object values.

The provided component instance can be a singleton or a new object can be returned on every invocation (prototype), this behavior is defined with the scope attribute, see Scope on page 263.

The provided object can optionally be notified when all of its properties have been injected, and when the providing bean manager will be deactivated, see Life Cycle Callbacks on page 265.

121.5 Bean Manager

The target type is normally used for listeners, service objects, and other places where a general application component instance is required.

121.5.1 Bean Component XML

The structure of a bean element is:

```
bean ::= ( <argument> | <property>)*
```

```
Figure 121.8 Bean Structure
```

121.5.2 <bean>

The Metadata for a bean manager is represented in the BeanMetadata interface, which extends ComponentMetadata. Table 121.1 provides an overview of the related XML definitions and the BeanMetadata interface. The table only provides a summary, the sometimes subtle interactions between the different features are discussed in later sections.

```
Table 121.1 Bean Manager Features
```

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Syntax</th>
<th>Bean Metadata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>ID</td>
<td>id</td>
<td>The id of a top level manager, must be unique in the Blueprint Container. All inlined managers must return null for their id.</td>
</tr>
<tr>
<td>Attribute or Element</td>
<td>Syntax</td>
<td>Bean Metadata</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>activation</td>
<td>lazy</td>
<td>activation</td>
<td>Defines if this bean is lazily or eagerly activated. If not explicitly set, the blueprint element's value for the default-activation attributes is used. If this is also not set, the value is eager. See Lazy and Eager on page 259.</td>
</tr>
<tr>
<td></td>
<td>eager</td>
<td>: int</td>
<td></td>
</tr>
<tr>
<td>depends-on</td>
<td>NCName*</td>
<td>dependsOn</td>
<td>Explicit list of ids that are the dependencies. These referred managers must be activated before this bean can provide an object. See Explicit Dependencies on page 259. This is a whitespace separated list.</td>
</tr>
<tr>
<td>class</td>
<td>qname</td>
<td>className</td>
<td>Class name of the object to be provided or the class name for a static factory. See Construction on page 264.</td>
</tr>
<tr>
<td>scope</td>
<td>singleton</td>
<td>scope</td>
<td>The scope defines the construction strategy for the component instance. The default is singleton except for inlined bean managers, where it is prototype. There is no schema default, so if it is not explicitly set, the Metadata will be null. See Scope on page 263.</td>
</tr>
<tr>
<td></td>
<td>prototype</td>
<td>: String</td>
<td></td>
</tr>
<tr>
<td>init-method</td>
<td>method</td>
<td>initMethod</td>
<td>The name of a method to invoke when a provided object has been injected with all its properties. If this is not set, it is null. See Life Cycle Callbacks on page 265.</td>
</tr>
<tr>
<td>destroy-method</td>
<td>method</td>
<td>destroyMethod</td>
<td>A name of a method to invoke on the provided objects with singleton scope when the Blueprint Container is destroyed. If this is not set, it is null. See Life Cycle Callbacks on page 265.</td>
</tr>
<tr>
<td>factory-method</td>
<td>method</td>
<td>factoryMethod</td>
<td>The name of the method on a static or component instance factory. See Construction on page 264.</td>
</tr>
<tr>
<td>factory-ref</td>
<td>NCName</td>
<td>factoryComponent</td>
<td>A reference to a manager that acts as the factory. See Construction on page 264.</td>
</tr>
</tbody>
</table>
The bean element has the following constraints that are not enforced by the schema but must be enforced by the Blueprint Container:

- The `destroyMethod` must not be set when the scope is `prototype`.
- The activation must not be set to `eager` if the bean also has `prototype` scope.
- The following combinations of arguments are valid, all other combinations are invalid:
  - `className`
  - `className`, `factory-method`
  - `factory-ref`, `factory-method`

### 121.5.3 `<argument>`

The argument element holds a value for a constructor or factory method's parameters.

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Syntax</th>
<th>Bean Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;argument&gt;</code></td>
<td>Table</td>
<td><code>arguments</code></td>
<td>Defined as sub-elements of the bean element. A BeanArgument object contains the value of an argument in the factory method or constructor. The order of the arguments is declaration order. See Construction on page 264.</td>
</tr>
<tr>
<td><code>&lt;property&gt;</code></td>
<td>Table</td>
<td><code>properties</code></td>
<td>Defined as sub-elements of the bean element. A BeanProperty object provides the property name and injection value. See Properties on page 264.</td>
</tr>
</tbody>
</table>

The `bean` element has the following additional constraints:

### Table 121.2 Bean Argument Features

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Syntax</th>
<th>Bean Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td><code>int &gt;= 0</code></td>
<td><code>index</code></td>
<td>The index of the argument in the constructor or factory-method signature. If this is not set, the Blueprint Container must use the type information to calculate it to match the disambiguation algorithm. The index will be -1 when not explicitly set.</td>
</tr>
<tr>
<td>type</td>
<td><code>qname</code></td>
<td><code>valueType</code></td>
<td>The fully qualified class name of a Java type to match the argument to the signature against.</td>
</tr>
<tr>
<td>ref</td>
<td><code>NCName</code></td>
<td><code>value</code></td>
<td>A reference to a top level manager that provides the value for the argument.</td>
</tr>
<tr>
<td>value</td>
<td><code>&lt;type&gt;&gt;</code></td>
<td><code>value</code></td>
<td>The Value Metadata based on the value property.</td>
</tr>
<tr>
<td></td>
<td>object</td>
<td><code>value</code></td>
<td>An inlined value.</td>
</tr>
</tbody>
</table>
- Either all arguments have a specified index or none have a specified index.
- If indexes are specified, they must be unique and run from 0..(n-1), where n is the number of arguments.
- The following attributes and elements are mutually exclusive:
  - ref
  - value
  - An inlined object value

121.5.4 <property>

The property element holds the information to inject a bean property with an object value.

Table 121.3 Bean Property Features

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Syntax</th>
<th>Bean Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>method</td>
<td>name</td>
<td>The property name, for example foo. The method name can consist of full stop separated method names, indicating nested property access.</td>
</tr>
<tr>
<td></td>
<td>( . method : String</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>)*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ref</td>
<td>NCName</td>
<td>value</td>
<td>A reference to a top level manager.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>: RefMetadata</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>&lt;&lt;type&gt;&gt;</td>
<td>value</td>
<td>A Value Metadata where the type is null.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>: ValueMetadata</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>: Metadata</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The argument element has the following additional constraints:

- The following attributes/elements are mutually exclusive
  - ref
  - value
  - An inlined object value

121.5.5 Scope

A bean manager has a recipe for the construction and injection of an object value. However, there can be different strategies in constructing its component instance, this strategy is reflected in the scope. The following scopes are architected for this specification:

- singleton - The bean manager only holds a single component instance. This object is created and set when the bean is activated. Subsequent requests must provide the same instance. Singleton is the default scope. It is usually used for core component instances as well as stateless services.
- prototype - The object is created and configured anew each time the bean is requested to provide a component instance, that is, every call to getComponentInstance must result in a new component instance. This is usually the only possible scope for stateful objects. All inlined beans are always prototype scope.

Implementations can provide additional scope types. However, these types must only be allowed when a defining namespace is included in the definitions and is actually used in the definitions to specify the dependency on this feature.
121.5.6 Construction

The Blueprint specification supports a number of ways for a bean manager to construct an object. Each possibility is a combination of the following Metadata properties:

- **className** - Defines the fully qualified name of a class to construct, or the name of a class with a static factory method. The class must be loadable from the Blueprint bundle `loadClass` method.
- **factoryMethod** - A static or instance factory method name that corresponds to a publicly accessible method on the given class or factory manager.
- **factoryComponent** - The id of a top-level target manager in the Blueprint Container that is an instance factory.

The Bean manager can have a number of `BeanArgument` objects that specify arguments for the constructor or for the factory class/object method. The matching constructor or method must be publicly accessible. The argument's `valueType` can be used to disambiguate between multiple signatures of constructors or methods. See *Signature Disambiguation* on page 285.

The value of the argument is always a `Metadata` object. Such an object can be converted into a general object value, see *Object Values* on page 279.

The construction properties can be used in a rather large number of combinations, however, not all combinations are valid. Table 121.4 shows the different valid combinations. If none of the combinations matches, then the Bean Metadata is erroneous.

In Table 121.4, a variation of the following bean definition is assumed:

```xml
<bean class="C" factory-method="f" factory-ref="fc">
  <argument value="1"/>
  <argument value="2"/>
</bean>
```

This definition is invalid because it specifies an invalid combination of metadata properties. The only valid combinations are subsets, they are all specified in the following table.

<table>
<thead>
<tr>
<th>className</th>
<th>factory-method</th>
<th>factory-ref</th>
<th>argument</th>
<th>Corresponding Java Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>f</td>
<td></td>
<td></td>
<td>new C</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>1,2</td>
<td>new C(1,2)</td>
</tr>
<tr>
<td>C</td>
<td>f</td>
<td>$fc</td>
<td></td>
<td>$fc.f()</td>
</tr>
<tr>
<td>f</td>
<td>$fc</td>
<td></td>
<td></td>
<td>$fc.f(1,2)</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>failure</td>
</tr>
</tbody>
</table>

The object created this way will be the provided object of the bean after any properties are injected. If the `factoryMethod` returns a primitive type, then this primitive must be converted to the corresponding wrapper type before any usage.

121.5.7 Properties

Dependency injection configures a constructed object with the help of the properties, which is a a List of `BeanProperty` objects. A Bean Property has the following features:

- **name** - The name of the bean property. This name refers to the set method on the constructed object as specified in the design pattern for beans getters and setters, see [5] *Java Beans Specification*. For example, if the property name is `foo`, then the public method `setFoo(arg)` will be used to set the value. There should only be one set method with a single argument for a specific property. If overloaded properties are encountered, the chosen set method is unspecified.
Nested property names are allowed when setting bean properties, as long as all parts of the path, except the property that is set, result in a non-null value. The parts of the path are separated with a full stop (\'\.\'\'). For example:

```xml
<property name="foo.bar.baz" value="42"/>
```

This example gets the foo property, from the constructed object, it then gets the bar property and then sets the baz property on that object with the given value.

- value - The value of the property is always a Metadata object. This Metadata object can be converted to a value object, see Object Values on page 279.

After the Metadata object is converted to an object value, it must be injected into the property. If the value object is not directly assignable to the property type (as defined by its only set method and the rules in Type Compatibility on page 286), then the Blueprint Container must use the type conversion mechanism to create a new object that matches the desired type, or fail. See Dependency Injection on page 285 for more information about dependency injection.

For example, the following bean creates an instance and then injects a three into a the foo property that it gets from the bar property. The string that holds the three is converted to a double:

```xml
<bean id="foo" class="com.acme.Foo">
    <property name="bar.foo" value="3"/>
</bean>
```

```java
// Classes
package com.acme;
public class Bar {
    double v;
    public void setFoo(double v) { this.v = v; }
}
public class Foo {
    Bar bar = new Bar();
    public void getBar() { return bar; }
}
```

```java
// Corresponding Java code
Foo foo = new Foo();
foo.getBar().setFoo(3.0);
```

### 121.5.8 Life Cycle Callbacks

The bean element provides two attributes that define the callback method names for initialization and destruction. A callback must be implemented as a publicly accessible method without any arguments. The callback method names must exist as void() methods.

The initMethod specifies the name of an initialization method that is called after all properties have been injected. The destroyMethod specifies the name of a destroy method that is called when the Blueprint Container has destroyed a component instance. Only bean managers with singleton scope support the destroyMethod. The destroy callback cannot be used for beans that have prototype scope, the responsibility for destroying those instances lies with the application.

### 121.5.9 Activation and Deactivation

A singleton bean manager must construct its single object during activation and then callback its initMethod method. Prototype scoped beans are created after activation and also have their initMethod invoked. The destroy method is called during the destruction of all the beans in singleton scope, this happens after deactivation.
A prototype bean manager has no special activities for deactivation.

121.6 Service Manager

The service manager defined by a service element is responsible for registering a service object with the service registry. It must ensure that this service is only registered when it is enabled. Where enabled means that all its mandatory service reference managers in its dependencies are satisfied.

121.6.1 <service>

The XML structure of the <service> manager is:

```
service ::= <interfaces>
         <service-properties>
         <registration-listener>*
target

interfaces ::= <value>+
service-properties ::= <entry>+
registration-listener ::= target
```

The service manager has the features outlined in Table 121.5 on page 266. The following additional constraints apply:

- The interface attribute and interfaces element are mutually exclusive.
- If the auto-export attribute is set to anything else but disabled, neither the interface attribute nor the interfaces element must be used.
- The ref attribute and inlined element are mutually exclusive.

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Type</th>
<th>Service Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>ID</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>: String</td>
<td></td>
</tr>
<tr>
<td>activation</td>
<td>lazy</td>
<td>activation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eager</td>
</tr>
<tr>
<td></td>
<td>: int</td>
<td></td>
</tr>
<tr>
<td>depends-on</td>
<td>NCName*</td>
<td>dependsOn</td>
</tr>
<tr>
<td></td>
<td>: List&lt;String&gt;</td>
<td></td>
</tr>
<tr>
<td>interface</td>
<td>qname</td>
<td>interfaces</td>
</tr>
<tr>
<td></td>
<td>: List&lt;String&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Table 121.5 Service Manager Features

- Optional component id of the manager, if it is a top level manager.
- Defines if this service is lazily or eagerly initialized. If not explicitly set, the blueprint element’s value for the default-activation attributes is used. If this is also not set, the value is eager. See also Lazy and Eager on page 259.
- Explicit list of ids that are the dependencies. These managers must be activated at the start of the registration phase. See Explicit Dependencies on page 259. This is a whitespace separated list.
- Name of the interface under which this service should be registered. See Service Interfaces on page 268.
<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Type</th>
<th>Service Metadata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto-export</td>
<td>disabled</td>
<td>autoExport</td>
<td>Defines the way the class must be analyzed to find the interfaces under which the service must be registered. The schema default is disabled. See Service Interfaces on page 268.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>interfaces</td>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class-hierarchy</td>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>all-classes</td>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ranking</td>
<td>int</td>
<td>ranking</td>
<td>The service.ranking value. The schema default is 0, which implies no service property. See Ranking on page 270.</td>
</tr>
<tr>
<td>ref</td>
<td>NCName</td>
<td>RefMetadata</td>
<td>Reference to the manager that provides the service object. See Service Object on page 270.</td>
</tr>
<tr>
<td>service-properties</td>
<td>List&lt;MapEntry&gt;</td>
<td>List&lt;String&gt;</td>
<td>The service properties for this service. See Service Properties on page 269.</td>
</tr>
<tr>
<td>registration-listeners</td>
<td>List&lt;RegistrationListener&gt;</td>
<td>List&lt;String&gt;</td>
<td>The registration listeners. See Registration Listener on page 270.</td>
</tr>
<tr>
<td>interfaces</td>
<td>List&lt;String&gt;</td>
<td>List&lt;String&gt;</td>
<td>Names of interfaces under which this service should be registered. Each interface name must be listed as a child value element. This value element has no attributes. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>Target</td>
<td></td>
<td>An inlined target manager that is used for the service object. See Service Object on page 270.</td>
</tr>
</tbody>
</table>

**121.6.2 `<registration-listener>`**

The `<registration-listener>` element can contain zero or more `<registration-listener>` elements, that define registration listeners to be notified of service registration and unregistration events. This element has the following structure:

```
registration-listener ::= target*  
```

The `<registration-listener>` element defines the callback methods for registration and unregistration.

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Type</th>
<th>Registration Listener</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref</td>
<td>NCName</td>
<td>Target</td>
<td>A reference to a top level manager.</td>
</tr>
<tr>
<td>Attribute or Element</td>
<td>Type</td>
<td>Registration Listener</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>registration-method</td>
<td>method</td>
<td>registrationMethod</td>
<td>The name of the method to call after the service has been registered. See Registration Listener on page 270.</td>
</tr>
<tr>
<td>unregistration-method</td>
<td>method</td>
<td>unregistrationMethod</td>
<td>The name of the method to call before the service will be unregistered. See Registration Listener on page 270.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;...&gt;</td>
<td>target</td>
<td>registrationListener</td>
<td>An inlined target manager</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The additional constraint is:

- The ref attribute and the inlined manager are mutually exclusive.
- Either or both of the registrationMethod and unregistrationMethod must be set.
- For each method name set, there must be at least one method matching the possible prototypes in the registration listener object, see Registration Listener on page 270.

121.6.3 Explicit Dependencies

A service manager must initialize any explicit dependencies in the start of its registration phase, even before it tracks its enabled state. The presence of explicit dependencies will not activate the service manager.

121.6.4 Provided Object

A service manager provides a proxy to a ServiceRegistration object. If this proxy is used when the dependencies are not met, and the service is therefore unregistered, an Illegal State Exception must be thrown. In all other cases, the proxy acts as if it was the ServiceRegistration object associated with the registration of its service object.

The unregister method on the returned object must not be used. If the application code calls unregister then this must result in an Unsupported Operation Exception.

121.6.5 Service Interfaces

Each service object is registered under one or more interface names. The list of interface names is provided by interfaces or autoExport.

The autoExport tells the Blueprint Container to calculate the interface(s) from the type of the service object. The autoExport can have the following values:

- disabled - No auto-detection of service interface names is undertaken, the interface names must be found in interfaces. This is the default mode.
- interfaces - The service object will be registered using all of its implemented public Java interface types, including any interfaces implemented by super classes.
- class-hierarchy - The service object will be registered using its actual type and any public super-types up to the Object class (not included).
- all-classes - The service object will be registered using its actual type, all public super-types up to the Object class (not including), as well as all public interfaces implemented by the service object and any of its super classes.

The autoExport requires the actual class object for introspection for all its modes except disabled, which can cause a bundle with a lazy activation policy to activate because a class will be loaded from the Blueprint bundle.
As an example:

```xml
<bean id="fooImpl" class="FooImpl"/>
```

```java
public class FooImpl implements Foo { ... }
```

Then the following service definitions are equivalent:

```xml
<service id="foo">
    <interfaces>
        <value>com.acme.Foo</value>
    </interfaces>
</service>

<service id="foo" interface="com.acme.Foo" ref="fooImpl"/>

<service id="foo" auto-export="interfaces" ref="fooImpl"/>
```

### 121.6.6 Service Properties

Each service can optionally be registered with *service properties*. The `serviceProperties` is a list of `MapEntry`, see `<entry>` on page 283. This metadata must be used to create the service properties. Service properties creation can have side effects because they can use component instances. The service properties must therefore be created once before the first time the service is registered.

The service manager adds the following automatic service properties that cannot be overridden. When these properties are explicitly set, they must be ignored.

- `osgi.service.blueprint.compname` - This will reflect the id of the manager that provides the service object, unless it is inlined. Inlined beans are always anonymous and must not have this property set.
- `service.ranking` - If the ranking attribute is not zero, this property will be set and hold an `Integer` object with the given value, see `Ranking` on page 270.

For example, the following definition is followed by equivalent Java code needed to register the service:

```xml
<service ref="fooImpl" interface="com.acme.Foo">
    <service-properties>
        <entry key="size" value="42"/>
    </service-properties>
</service>
```

```java
Dictionary d = new Hashtable();
d.put("size", "42");
d.put("osgi.service.blueprint.compname", "fooImpl");
ServiceRegistration sr =
    bundleContext.registerService("com.acme.Foo",
        blueprintContainer.getComponentInstance("fooImpl"),
        d);
```

Service properties should specify the `valueType` of the entry unless the value to be registered needs to be a `String` object. The service property types should be one of:

- **Primitives Number** - `int`, `long`, `float`, `double`, `byte`, `short`, `char`, `boolean`
- **Scalar** - `String`, `Integer`, `Long`, `Float`, `Double`, `Byte`, `Short`, `Character`, `Boolean`.
- **Array** - An array of either the allowable primitive or scalar types.
- **Collection** - An object implementing the Collection interface that contains scalar types.
See `<entry>` on page 283 types for information how to create these types.

### 121.6.7 Service Object

The service manager must not request the Blueprint Container for the service object until it is actually needed because a bundle requests it. The service object is represented in the value. This is a Metadata object that can be used to construct an object value, see *Object Values* on page 279.

For example:

```
<service id="fooService" ref="fooImpl".../>
```

```
<service id="fooService" ... >
    <bean class="com.acme.fooImpl"/>
</service>
```

The scope of the beans is ignored for the manager that provides the service object. Its value will only be created once the first time it is needed for the service.

### 121.6.8 Scope

A service manager must always register a Service Factory as service object and then dispatch the service requests to the service object. A service manager must obtain a single component instance as service object. This component instance is shared between all bundles. That is, even if the service object comes from a prototype scoped manager, only one instance is ever created per service manager.

If this component instance implements Service Factory, then all incoming service requests are forwarded to this single component instance.

### 121.6.9 Ranking

When registering a service with the service registry, an optional *service ranking* can be specified that orders service references. The service ranking is registered as the SERVICE_RANKING property defined in the OSGi service layer. When a bundle looks up a service in the service registry, given two or more matching services, then the one with the highest number will be returned. The default ranking value for the OSGi service registry is zero, therefore, this property must not be registered when ranking is zero, which is also the default value.

For example:

```
<service ref="fooImpl" interface="com.acme.FooImpl"
    ranking="900"/>
```

This will result in the following service property:

```
service.ranking=new Integer(900)
```

### 121.6.10 Registration Listener

The registrationListeners represent the objects that need to be called back after the service has been registered and just before it will be unregistered.

The listenerComponent must be a Target object; it is the target for the following callbacks:

- *registrationMethod* · The name of the notification method that is called after this service has been registered.
- *unregistrationMethod* · This method is called when this service will be unregistered.

The signatures for the callback methods depend on the scope and if the service object implements the ServiceFactory interface. The different possibilities are outlined in the following table.
**Table 121.7 Interaction scopes and types for callback signature.**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Type</th>
<th>Signature</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>singleton</td>
<td>ServiceFactory</td>
<td>void(ServiceFactory,Map)</td>
<td>All service requests are handled by the component instance.</td>
</tr>
<tr>
<td>singleton</td>
<td>T</td>
<td>void( super T,Map)</td>
<td>T is assignable from the service object's type.</td>
</tr>
<tr>
<td>prototype</td>
<td>ServiceFactory</td>
<td>void(ServiceFactory,Map)</td>
<td>All service requests are handled by the first component instance.</td>
</tr>
<tr>
<td>prototype</td>
<td>T</td>
<td>void(,Map)</td>
<td>The first argument must be null because for prototype service objects, the component instance is created when a bundle requests the service. Therefore, at registration time there is no service object available.</td>
</tr>
</tbody>
</table>

If multiple signatures match, then all methods must be called in indeterminate order. At least one method must match.

The service manager must provide the registration listener with the current registration state when the listener is registered. This initial notification must take place before any other callback methods are called on this listener on other threads. That is, if the service is registered at that time, it must call the registration method and otherwise the unregistration method.

The following example shows two registration listeners, one with a referred bean and another one with an inlined bean.

```xml
<service ref="fooImpl" interface="com.acme.Foo">
  <registration-listener registration-method="reg" unregistration-method="unreg">
    <bean class="com.acme.FooListener"/>
  </registration-listener>
</service>

<service ref="fooImpl" interface="com.acme.Foo">
  <registration-listener registration-method="reg" unregistration-method="unreg" ref="fooListener"/>
</service>

<bean id="fooListener" class="com.acme.FooListener"/>
```

```java
package com.acme;
public class FooListener {
  public void reg( Foo foo, Map properties ) { ... }
  public void unreg( Foo foo, Map properties ) { ... }
}
```

The manager that provides the registration listener object is an implicit dependency of the enclosing service manager. However, the registration listener component instance is specifically allowed to use to the service manager though this is technically a cyclic dependency. Therefore, a bean is allowed to be both be injected with a ServiceRegistration object from the service manager as well as being a registered listener to the same service manager.

In the following example, the foo service manager uses manager main, both as a registration listener as well as top-level bean main being injected with reference foo.
A service manager needs a service object that is referred to by the value Metadata property. This value can in its turn depend on other managers transitively. If any of these managers are service reference managers, then they can be satisfied or not. If these service reference managers are marked to be mandatory, then they influence the enabled state of the first service manager. Only if all of these mandatory service reference managers in the dependency graph are satisfied, then the first service manager is enabled.

A service manager must have a Service Factory registered with the OSGi service registry after the primary initialization of the Blueprint Container has been done until the Blueprint Container is destroyed while it is enabled. See see Service Registration on page 253.

121.6.12 Activation and Deactivation

When a service manager is activated, it must actuate its registration listeners. Each registration listener must be called back during its actuation with the current service registration state as described in the Registration Listener on page 270. Normally, this will also request the container for a service object but this can be further delayed in certain circumstances. See Service Object on page 270 for more details.

During deactivation, a service manager must disable any registration listeners and release any dependencies it has on these component instances.

121.7 Service Reference Managers

The reference, and reference-list elements are all service references. They select a number of services in the service registry. The structure of these elements is as follows:

```
reference ::= <reference-listener>*
reference-list ::= <reference-listener>*
```

The inheritance hierarchy for service references is depicted in Figure 121.9 on page 272.
121.7.1 Service Reference

The service reference managers have almost identical Metadata and share most behavior. The only schema differences between a reference manager and a reference-list manager are:

- **timeout** - A reference manager supports a timeout.
- **memberType** - The reference-list can define its member-type.

The features of the service references are explained in the following table.

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Type</th>
<th>ServiceReference-Metadata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>ID</td>
<td>id</td>
<td>The component id of a top level manager</td>
</tr>
<tr>
<td>activation</td>
<td>lazy</td>
<td>activation</td>
<td>Defines if this service reference is lazily of eagerly initialized. If not explicitly set, the blueprint element's value for the default-activation attributes is used. If this is also not set, the value is eager. See also Lazy and Eager on page 259.</td>
</tr>
<tr>
<td></td>
<td>eager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>depends-on</td>
<td>NCName*</td>
<td>dependsOn</td>
<td>Explicit list of component ids that are the dependencies. These managers must be activated before this service reference's activation. See Explicit Dependencies on page 259. This is a whitespace separated List.</td>
</tr>
<tr>
<td>availability</td>
<td>mandatory</td>
<td>availability</td>
<td>Defines if a service reference is mandatory or optional. The default for the availability attribute is defined by the default-availability attribute in the blueprint element. If the default-availability attribute is not defined, the value is mandatory.</td>
</tr>
<tr>
<td></td>
<td>optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interface</td>
<td>qname</td>
<td>interface</td>
<td>A single name of an interface class. It is allowed to not specify an interface name.</td>
</tr>
<tr>
<td>component-name</td>
<td>NCName</td>
<td>componentName</td>
<td>Points to another manager in another Blueprint Container registered in the service registry. If set, the component name must be part of the effective filter.</td>
</tr>
<tr>
<td>filter</td>
<td>filter</td>
<td>filter</td>
<td>The given filter string, can be null.</td>
</tr>
</tbody>
</table>

The features of the service references are explained in the following table.

Table 121.8 Service Reference Manager Features
<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Type</th>
<th>ServiceReference-Meta-data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;reference-listener&gt;</td>
<td>See &lt;reference-listener&gt; on page 274</td>
<td>referenceListeners : List&lt;Listener&gt;</td>
<td>The Metadata of the reference listeners</td>
</tr>
</tbody>
</table>

The additional constraints for service references are:

- The interface, if set, must refer to a public interface.

### 121.7.2 <reference>

A reference manager, selecting a single service, has the additional feature explained in the following table.

Table 121.9 Reference Features

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Type</th>
<th>Reference Metadata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>long &gt;= 0</td>
<td>timeout : long</td>
<td>The timeout in ms. Zero is indefinite.</td>
</tr>
</tbody>
</table>

An additional constraint on the reference is:

- The timeout must be equal or larger than zero.

### 121.7.3 <reference-list>

A reference-list manager, selecting multiple services, has the additional feature explained in the following table.

Table 121.10 Reference-list Features

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Type</th>
<th>Reference List Metadata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>member-type</td>
<td>service-object</td>
<td>memberType : int</td>
<td>Defines if the members of the list are ServiceReference objects or the proxies to the actual service objects.</td>
</tr>
<tr>
<td></td>
<td>service-reference</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 121.7.4 <reference-listener>

The reference element can notify reference listeners of the service selection changes with the reference-listeners. The reference-listener element has the following structure:

reference-listener ::= target*

The reference-listener element defines the callback methods for binding and unbinding a service.

Table 121.11 Reference Listener Features

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Type</th>
<th>Reference Listener</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref</td>
<td>NCName</td>
<td>listenerComponent</td>
<td>A reference to a top level target manager.</td>
</tr>
<tr>
<td>bind-method</td>
<td>method</td>
<td>bindMethod : String</td>
<td>The name of the method to call after the service has been bound. See Reference Listeners on page 277.</td>
</tr>
<tr>
<td>Attribute or Element</td>
<td>Type</td>
<td>Reference Listener</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>unbind-method</td>
<td>method</td>
<td>unbindMethod</td>
<td>The name of the method to call before the service will be unbound. See Reference Listeners on page 277.</td>
</tr>
<tr>
<td>&lt;...&gt;</td>
<td>target</td>
<td>listenerComponent</td>
<td>An inlined target manager</td>
</tr>
</tbody>
</table>

The additional constraints are:

- The ref attribute and the inlined manager are mutually exclusive.
- Either or both bindMethod and unbindMethod must be specified.
- At least one specified method must exist with each given method name, see Reference Listeners on page 277.

121.7.5 Provided Object For a Reference

The provided object for a service reference manager is a proxy backed by a service object from the service registry. Therefore, even though the injected object will remain constant, it can change its reference to a backing service at any time, implying it can only be used with stateful services if reference listeners are used. If use when no suitable backing service is available, it will wait until it times out. See Service Dynamics on page 292 for more details. The model is depicted in Figure 121.10.

![Figure 121.10 Constant references with dynamic selection](image)

The following example shows how a property can be set to the service object.

```java
class C {
    public void setProxy(T ref) { ... }
}
<reference id="p" interface="T"/>
<bean id="c" class="C">
    <property name="proxy" ref="p"/>
</bean>
```

121.7.6 Provided Object For a Reference-list

The reference-list provided object implements the List interface; this List contains proxies to the backing services. These proxies do not have a timeout. That is, when a proxy from a reference-list is used, it must not wait when the backing service is no longer available but it must immediately throw a Service Unavailable Exception.

Changes to the list are dynamic. When a backing service is unregistered, the corresponding proxy is removed from the list synchronously with the service event. When a new service enters the selection, it is added synchronously with the service event. Proxies to newly discovered services must be added at the end of the list. The structure is depicted in Figure 121.11.
The member type of the list depends on the `memberType`. If this is set to:

- `service-object` - Inject a List of service objects, this is the default.
- `service-reference` - Inject a list of `ServiceReference` objects

If generics information is available, then it is an error if the generic member type of the target list is not assignable with the `memberType`. If the member target type is in itself specified with generic arguments, like `List<T<U>>`, then the assignment must fail because this would require conversion and no conversion can take place for this assignment. For information about generics, see `Generics` on page 291.

### 121.7.7 Read Only Lists

The list is a read-only view on the actual set of proxies to the service objects. This `List` object must only support the following methods:

- `contains(Object)`
- `containsAll(Collection)`
- `equals(Object)`
- `get(int)`
- `hashCode()`
- `indexOf(Object)`
- `isEmpty()`
- `iterator()` // no remove method
- `lastIndexOf(Object)` // not supported
- `listIterator()` // not supported
- `listIterator(int)`
- `size()`
- `subList(int, int)` // same list type as parent
- `toArray()`
- `toArray(T[])`

All other methods must throw an Unsupported Operation Exception. The List Iterator is not supported for these lists.

### 121.7.8 Selection

A service reference must provide a `selection` of services from the service registry. The Blueprint Container must logically use a filter for the selection that is the and (`&`) of the following assertions:

- The interface, if specified
- If `componentName` is not null, a filter that asserts `osgi.blueprint.compname=$componentName`
  
  This is a convenience function to easily refer to managers in other Blueprint Containers. Registered Blueprint services will automatically get this property set to their blueprint name.
- If `filter` is not null, the filter
The selection is defined as the set of Service References selected by the given filter.

121.7.9 Availability

A service reference is satisfied when one or more services match the selection. The availability is used to specify whether a service reference needs to be satisfied before initialization, see Grace Period on page 253, or if it controls the registration state of any service managers that depend on this service reference manager (explicit and implicit), see Mandatory Dependencies on page 294. The availability can have the following values:

- mandatory - Mandatory indicates that the service reference needs to be satisfied.
- optional - Optional indicates that the satisfaction of this reference is not relevant for any registered services, or for the grace period.

It is an error to declare a mandatory reference to a service that is registered by the same bundle. Such a definition could cause either deadlock or a timeout.

The fact that Blueprint specification has mandatory service references gives no guarantee that a valid service object is available when the service reference is used, in the dynamic world of OSGi, services can get unregistered at any time.

The following example declares a mandatory service reference for a single service. The usage of the reference can stall a maximum of 5 seconds if no service matches the selection.

```xml
<reference
  id          = "log"
  interface   = "org.osgi.service.log.LogService"
  availability="mandatory"
  timeout     = "5000" />
```

121.7.10 Reference Listeners

The reference listeners are represented as ReferenceListener objects. They define the following call-backs:

- bindMethod - Called after a service is selected by the service reference manager. For a reference manager, this method can be called repeatedly without an intermediate unbind callback. This happens when a service is unregistered but a replacement can be found immediately.
- unbindMethod - Called when the service is no longer used by the service reference manager but before it has been returned to the service registry with the unget method. For a reference manager, no unbind method is called when the service can immediately be replaced with an alternative service when the service goes away.

A reference listener callback can have any of the following signatures:

- public void(ServiceReference) - Provide the ServiceReference object associated with this service reference. This callback type provides access to the service's properties without actually getting the service.
- public void( super T) - Provide the proxy to the service object, where T is on of the types implemented by the service object proxy.
- public void( super T,Map) - Provide the proxy to the service object. T is a type that is assignable from the service object. The Map object provides the service properties of the corresponding ServiceReference object.

All signatures must be supported regardless of the value of memberType that was specified in the reference-list. The service object given to the reference listeners must be the proxy to the service object.
The callbacks must be made synchronously with the corresponding OSGi service event. For reference-list callbacks, the service proxy is guaranteed to be available in the collection before a bind callback is invoked, and to remain in the collection until after an unbind callback has completed.

If a service listener defines multiple overloaded methods for a callback, then every method with a matching signature is invoked in an undefined order.

For example, the following definition will result in calling all the setLog methods on a FooImpl object:

```xml
<reference id="log" interface="org.osgi.service.log.LogService">
  <reference-listener bind-method="setLog">
    <bean class="com.acme.FooImpl"/>
  </reference-listener>
</reference>

public class FooImpl {
  public void setLog(Object o, Map m) { ... }
  public void setLog(LogService l, Map m) { ... }
  public void setLog(ServiceReference ref) { ... }
}
```

The manager that provides the reference listener object is treated as an implicit dependency of the enclosing service reference. This manager is specifically allowed to use the service reference in a property injection or constructor argument, though this is technically a cyclic dependency. Therefore, a bean must be allowed to both be injected with a reference as well as listening to the bind and unbind callbacks of that same reference.

In the following example, the foo reference manager uses manager main, both as a reference listener as well as manager main being injected with reference foo.

```xml
<reference id="foo" interface="com.acme.Foo">
  <reference-listener bind-method="setL" ref="main"/>
</reference>
<bean id="main" class="com.acme.Main">
  <property name="r" ref="foo"/>
</bean>
```

121.7.11 Service Proxies

The Blueprint extender must generate proxies for the service reference managers. Reference managers provide proxies that dynamically select a backing service, which can change over time. A reference-list provides a list of proxies that have a fixed backing service, these proxies are added and removed from the list, based on the selection, they do not have a time-out.

The backing service for a reference proxy must not be gotten from the OSGi service registry until an actual service object is needed, that is, when an actual method is called on the proxy. If the backing service becomes unregistered, then the proxy must unget the reference to the backing service (if it had gotten it) and get another service object the next time a method on the proxy is called. If a replacement can be found immediately, the reference listener’s bind method must be called without calling the unbind method. Other threads that need the same service object must block until the service object has become available or times out.

The proxies must implement all the methods that are defined in the interface. The interface must refer to an interface, not a class. The proxy must only support the methods in the given interface. That is, it must not proxy methods available on the service object that are not available in the given
interface. If no interface is defined, the proxy must be implemented as if the interface had no methods defined.

Blueprint bundles must ensure that the proper semantics are maintained for `hashCode` and `equals` methods. If these methods are not defined in the interface, then the proxy must use the default semantics of the `Object` class for `equals` and `hashCode` methods.

### 121.7.12 Activation and Deactivation

Service reference managers are active before activation because they must handle the enable status of service managers.

During activation, a service reference must actuate its listeners and provide these listeners with the initial state of the reference. For a reference, if there is a selected object, the `bind` method must be called with the proxy object, otherwise the `unbind` method must be called with a `null` as proxy object. For a reference-list, the `bind` method must be called for each member of the list. If the list is empty, the `unbind` method must be called with a `null` as proxy object.

During deactivation, the listeners must be disabled.

### 121.8 Object Values

Top-level managers can use *object values* in different places. These object values are defined with XML elements and attributes. After parsing, they are all converted to sub-interfaces of the `Metadata` interface, transitively reachable from top-level managers. For example, the following definition creates a bean that is injected with the byte array: `byte[] {7, 42}:

```
<bean class="com.acme.FooImpl">
  <property name="array">
    <array value-type="byte">
      <value>7</value>
      <value>42</value>
    </array>
  </property>
</bean>
```

This definition provides the configuration data for an *array value*, which is represented by the `CollectionMetadata` interface. A `Metadata` object can be used to construct its object value during runtime whenever a new object must be constructed.

In most places where an object value can be used, it can be anything, including objects provided by a manager and even `null`. However, maps require non-null keys. The object values are therefore split in `value` and `nonNullValue` types.

The syntax for object values has the following structure:

```
nonNullValue ::= <ref>
  | <idref>
  | <value>
  | <map>
  | <props>
  | <list>
  | <set>
  | <array>
```

Object values also include inlined managers. The use of an inlined manager for an object value means that manager will provide a value every time the object value is constructed. Each of the ob-
Object values is created anew and the types are mutable, except for the service references. The use of managers in object values must create an implicit dependency between the top level managers and any transitively reachable manager from their Metadata.

### 121.8.1 <ref>

The ref element is a reference to a top-level manager in the same Blueprint Container. The ref element has a single attribute component-id.

**Table 121.12 Ref Features**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Ref Metadata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component-id</td>
<td>NCName</td>
<td>componentId</td>
<td>A reference to a top level manager.</td>
</tr>
</tbody>
</table>

For example, the following definition uses the foo manager to instantiate the service object.

```xml
<service id="fooService" interface="com.acme.Foo">
  <ref component-id="fooImpl"/>
</service>
```

```java
public class FooImpl implements Foo { }
```

### 121.8.2 <idref>

The idref element provides the component id of another manager in the same Blueprint Container. This reference can then be used by the application to look up a manager in the Blueprint Container during runtime. The idref element is a safe way to provide a component id because the Blueprint Container will verify that the component id exists, thereby showing errors early. The idref does not create an implicit dependency on the given manager.

**Table 121.13 IdRef Features**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Id Ref Metadata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component-id</td>
<td>NCName</td>
<td>componentId</td>
<td>A reference to a top level manager.</td>
</tr>
</tbody>
</table>

The following example provides the foo object with the reference to the database.

```xml
<bean id="foo" class="com.acme.FooImpl">
  <property name="db">
    <idref component-id="jdbc"/>
  </property>
</bean>
```

```java
<bean id="jdbc" ... />
```

The following definition is equivalent to except that a non existent component id will not be detected until the foo object access the Blueprint Container. In the previous example this was detected directly after the definitions were parsed.

```xml
<bean id="foo" class="com.acme.FooImpl">
  <property name="db" value="jdbc"/>
</bean>
```
A value element represents an object that can directly be constructed from a string formed by its text contents.

Table 121.14  Value Features

<table>
<thead>
<tr>
<th>Attribute, Element</th>
<th>Type</th>
<th>Value Metadata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>type</td>
<td>: String</td>
<td>The optional type name to be used in type converting the given string to a target type. This type can commit the conversion to a specific choice. If this type is not set, then it must return null. For the type syntax, see Syntax for Java types on page 256.</td>
</tr>
<tr>
<td>&lt;&lt;type&gt;&gt;</td>
<td>stringValue</td>
<td>: String</td>
<td>The string value that must be converted to the target type, if set.</td>
</tr>
</tbody>
</table>

If a value element is used as a member in a list, map, array, or set then the enclosing collection can define a default value for the type attribute of its value elements.

The following example creates a list of two OSGi version objects.

```xml
<list value-type="org.osgi.framework.Version">
  <value>1.3.4</value>
  <value>5.6.2.v200911121020</value>
</list>
```

The corresponding Java code is:

```java
Arrays.asList( new Version("1.3.4"),
               new Version("5.6.2.v200911121020") )
```

A null element results in a Java null. It has no attributes and no elements. It corresponds to Null Metadata.

Lists, sets, and arrays are referred to as collections. List and array are ordered sequences of objects, where equal objects can occur multiple times. A set discards equal objects.

The structure of a collection element is:

```java
collection ::= value *
```

Table 121.15  Collection Features

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Type</th>
<th>Collection Metadata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value-type</td>
<td>type</td>
<td>: String</td>
<td>Optionally set the type for ValueMetadata children.</td>
</tr>
</tbody>
</table>
### Table 121.16 Map Features

<table>
<thead>
<tr>
<th>Attribute or Element</th>
<th>Type</th>
<th>Map Metadata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key-type</td>
<td>type</td>
<td>keyType</td>
<td>Optional default type for keys. For the syntax see Syntax for Java types on page 256.</td>
</tr>
<tr>
<td>value-type</td>
<td>type</td>
<td>valueType</td>
<td>Optional default type for values. For the syntax see Syntax for Java types on page 256.</td>
</tr>
<tr>
<td>&lt;entry&gt;</td>
<td></td>
<td>values</td>
<td>The MapEntry object for the children of the map or properties.</td>
</tr>
</tbody>
</table>

There are no additional constraints.

The valueType sets the default for any contained ValueMetadata objects. The result of a collection element is an object that implements the given collection interface or is an Object[]. That is, the resulting object is mutable and can be used by the application. However, type conversion can create a copy of this list.

The following example creates a List of Lists of 2x2 of int values:

```xml
<list>
  <list value-type="int">
    <value>2</value>
    <value>7</value>
  </list>
  <list value-type="int">
    <value>9</value>
    <value>5</value>
  </list>
</list>
```

The corresponding Java code is:

```java
Arrays.asList(
    new int[]{2,7},
    new int[]{9,5});
```

### 121.8.6 `<map>`

A map is a sequence of associations between a `<key>` and some object, this association is called an `<entry>`. The structure of a map element is therefore:

```
map ::= <entry> *
```

The valueType sets the default for the children of the collection

The Metadata for the children of the collection

The actual collection class to be used, derived from the appropriate definition.

The corresponding Java code is:

```java
Arrays.asList(
    new int[]{2,7},
    new int[]{9,5});
```
121.8.7  <entry>

The entry element provides an association between a key and a value. The structure of the element is:

\[
\text{entry} ::= \text{<key>} \ \text{object} \\
\text{key} ::= \text{nonnullValue}
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Map Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>&lt;&lt;type&gt;&gt;</td>
<td>key</td>
<td>Specify the key of the entry.</td>
</tr>
<tr>
<td>key-ref</td>
<td>NCName</td>
<td>key</td>
<td>Reference to a top-level manager</td>
</tr>
<tr>
<td>&lt;key&gt;</td>
<td>nonNull-Value</td>
<td>key</td>
<td>Contains an inlined value that is never null.</td>
</tr>
<tr>
<td>value</td>
<td>&lt;&lt;type&gt;&gt;</td>
<td>value</td>
<td>Specify the value directly, this will be a string type.</td>
</tr>
<tr>
<td>value-ref</td>
<td>NCName</td>
<td>value</td>
<td>A reference to a top-level manager</td>
</tr>
<tr>
<td>&lt;...&gt;</td>
<td>object</td>
<td>value</td>
<td>An inlined manager</td>
</tr>
</tbody>
</table>

Additional constraints:
- key, key-ref attributes and key element are mutually exclusive.
- value, value-ref attributes and value element are mutually exclusive.
- The resulting object of a key must not be a primitive type.

The following example shows the different way an entry can get its key. In this case the value is always a string.

```xml
<map>
  <entry key="bar" value="..."/>    // 1
  <entry key-ref="bar" value="..."/>    // 2
  <entry value="..."/>
    <key>
      <value type="org.osgi.framework.Version">2.71</value>
    </key>
  </entry>
</map>
```

The previous example is equivalent to the following Java code:

```java
Map m = new HashMap();
m.put( "bar", "..." );
m.put( container.getComponentInstance( "bar" ), "..." );
m.put( new Version( "2.71" ), "..." );
```

The following examples shows the different ways a value of an entry can be defined.
<map>
  <entry key="1" value="1"/>
  <entry key="2" value-ref="foo"/>
  <entry key="3">
    <value type="org.osgi.framework.Version">3.14</value>
  </entry>
</map>

The previous code is equivalent to the following Java code:

```java
Map m = new HashMap()
m.put("1", "1");
m.put("2", container.getComponentInstance("foo"))
m.put("3", new Version("3.14"));
```

121.8.8

The `props` element specifies a `Properties` object. The structure of a `props` element is as follows:

```
props ::= prop *
```

Each `prop` element is an association between two strings. It defines the following attributes:

- `key` - A string specifying the property key. This attribute is required.
- `value` - A string specifying the property value.

The following example initializes the same `Properties` object in two ways:

```xml
<props>
  <prop key="1" value="one"/>
  <prop key="2" value="two"/>
</props>
<props>
  <prop key="1" value="one"/>
  <prop key="2" value="two"/>
</props>
```

This is equivalent to the following Java code:

```java
Properties p = new Properties();
p.setProperty("1", "one");
p.setProperty("2", "two");
```

121.8.9

Manager as Value

Each manager can be the provider of component instances that act as object values. When a manager is used in an object value, then that is the manager asked to provide a component instance. The managers are specified in `manager` on page 258. The simple example is a bean. Any inlined bean can act as an object value. For example:

```xml
<list>
  <bean class="com.acme.FooImpl" />
</list>
```

Some managers have side effects when they are instantiated. For example, a service manager will result in a `ServiceRegistration` object but it will also register a service.
**121.9 Dependency Injection**

A bean has a recipe for constructing a component instance with a constructor or factory and then providing it with its properties. These properties are then injected with object values, see Object Values on page 279.

The following types of dependencies can be injected:

- **Constructor arguments** - The arguments specify the parameters for a constructor.
- **Static Factory arguments** - The arguments specify the parameters for a static method.
- **Instance Factory arguments** - The arguments specify the parameters for a method on an object provided by another manager.
- **Properties** - The value of the Bean Property specifies the single parameter for the property's set method.

In all the previous cases, the Blueprint Container must find an appropriate method or constructor to inject the dependent objects into the bean. The process of selecting the correct method or constructor is described in the following section, which assumes a Bean Argument as context, where a Bean Property acts as a Bean Argument without an index or type set.

### 121.9.1 Signature Disambiguation

Constructors, factory methods, and property set methods are described with Metadata. The Blueprint Container must map these descriptions to an actual method or constructor. In practice, there can be multiple methods/constructors that could potentially map to the same description. It is therefore necessary to disambiguate this selection. Both factory methods and constructors have the same concept of signatures. A signature consists of an ordered sequence of zero or more types. For methods, only publicly accessible methods with the appropriate name are considered. For constructors, all publicly accessible constructors are considered. The disambiguation process described here is valid for all constructors and methods because the signature concept applies to both of them.

1. Discard any signatures that have the wrong cardinality.
2. Find the list of signatures that have assignable types for each argument in their corresponding positions. Assignable is defined in Type Compatibility on page 286. If a type was specified for an argument, then this type must match the name of the corresponding reified type in the signature exactly.
3. If this result list has one element, then this element is the answer. If this list has more than one element, then the disambiguation fails.
4. Otherwise, find the list of signatures that have compatible types for each argument in their corresponding positions. Compatibility is defined in Type Compatibility on page 286.
5. If this result list has one element, then this element is the answer. If the list has more than one element, then the disambiguation fails.
6. If the arguments cannot be reordered (the index of the argument is used and is thus not -1, or there are less than two arguments) then the disambiguation fails.
7. Find all signatures that match a re-ordered combination of the arguments. Reordering must begin with the first argument and match this argument against the first assignable types in a sig-
nature, going from position 0 to n. If the type is assignable from the argument, then it is locked in that position. If the argument has a type, then it must exactly match the name of the selected signature type. The same is done for the subsequent arguments. If all arguments can find an exclusive position in the signature this way, then the signature is added to the result.

8. If the result list contains one signature, then this is the resulting signature. If the list has more than one element, then the disambiguation fails.

9. Repeat step 6, but now look for compatible types instead of assignable types.

10. If the result list contains one signature, then this is the resulting signature.

11. Otherwise, the disambiguation fails

An example elucidates how the disambiguation works. Assuming the following definition and classes:

```xml
<bean ...
  <argument>
    <bean class="Bar"/>
  </argument>
  <argument>
    <bean class="Foo"/>
  </argument>
</bean>
```

```java
public class Bar extends Foo {}
public class Foo {}
```

The following bullets provide examples how signatures are matched against the previous definition.

- (Bar,Foo) - The arguments will be in the given order and the ordered match will succeed. This is the normal case.
- (Foo,Bar) - This will not match because in the re-ordered match, the Bar argument (which is a Foo sub-type) is matched against the first argument. The second Foo argument can then no longer find a compatible type because that slot is taken by the Bar instance.
- (Object,Object) - This will be called with (aBar,aFoo).

Multiple constructors on a class can require disambiguation with the arguments type. In the following example, the Multiple class has two constructors that would both match the constructor arguments because a String object can be converted to both a File object and a URL object.

```java
public class Multiple {
    public Multiple(URL a);
    public Multiple(File a);
}
```

An attempt to configure a Multiple object without the type will fail, because it is not possible to determine the correct constructor. Therefore, the type should be set to disambiguate this:

```xml
<bean class="Multiple">
  <argument type="java.net.URL" value="http://www.acme.us"/>
</bean>
```

### 121.9.2 Type Compatibility

During injection, it is necessary to decide about type assignability or type compatibility in several places. If generics are present, a type must be reified in its class, see Generics on page 291. In this specification, the canonical representation for a type is T<P1..Pn>, where n is zero for a non-parame-
terized type, which is always true in a VM less than Java 5. The `ReifiedType` class models this kind of type.

If type `T` or `S` is primitive, then they are treated as their corresponding wrapper class for deciding assignability and compatibility. Therefore, a type `T<P1..Pn>` (target) is assignable from an object `s` of type `S` (source) when the following is true:

- `n == 0`, and
- `T assignableFrom(S)

`T<P1..Pn>` is compatible with an object `s` of type `S` when it is assignable or it can be converted using the Blueprint built-in type converter. The convertability must be verified with the `canConvert(s, T<P1..Pn>)` method. That is, type compatibility is defined as:

- `assignable(T<P1..Pn>, S)`, and
- `cs.canConvert(s, T<P1..Pn>)` returns `true`

Where `cs` is the Blueprint built-in type converter that also uses the custom type converters.

### 121.9.3 Type Conversion

Strings in Blueprint definitions, object values, and component instances must be made compatible with the type expected by an injection target (method or constructor argument, or property) before being injected, which can require type conversion. The Blueprint Container supports a number of built-in type conversions, and provides an extension mechanism for configuring additional type converters. Custom type converters have priority over built-in converters.

The goal of the type conversion is to convert a source object `s` with type `S` to a target type `T<P1..Pn>`. The conversion of the Blueprint built-in type converter must take place in the following order:

1. If `T<P1..Pn>` is assignable from `S`, which implies `n=0`, then no conversion is necessary, except that primitives must be converted to their wrapper types.
2. Try all type converters in declaration order with the `canConvert(s, T<P1..Pn>)` method, exceptions are ignored and logged. The first converter that returns `true` is considered the converter, its result is obtained by calling `convert(s, T<P1..Pn>)`. Exceptions in this method must be treated as an error.
3. If `T` is an array, then `S` must be an array or it must implement Collection, otherwise the conversion fails. Each member of array `s` must be type converted to the component type of `T` using the generics information if available, see the `getComponentType` method on `Class`. This is a recursive process. The result must be stored in an array of type `T`.
4. If `T` implements Collection, then `S` must be an array or implement Collection, otherwise the conversion fails. If the platform supports generics, the members of object `s` must be converted to the member type of the collection if this is available from the generics information, or to `Object` otherwise. The Blueprint Container must create a target collection and add all members of `s` to this new object in the iteration order of `s`. The target collection depends on type `T`:
   - If `T` is one of the interfaces listed in Concrete Types for Interfaces on page 290, then the target collection must be the corresponding concrete class.
   - `T` must represent a public concrete class with an empty publicly accessible constructor, the target collection is then a new instance of `T`.
   - Otherwise `T` represents an interface and the conversion must fail.
5. If `T` implements Map or extends Dictionary, then `S` must implement Map or extend Dictionary as well, otherwise the conversion fails. If the platform supports generics, the members of map `s` must be converted to the key and value type of the target map. This is a recursive process. Without generics, the members are not converted and put as is.

The target map depends on `T:`
• If \( T \) is a public concrete class (not interface) with an empty publicly accessible constructor then the target map must be a new instance of \( T \).

• If \( T \) is one of the Map interfaces or Dictionary listed in Concrete Types for Interfaces on page 290, then the target map must be the corresponding concrete class.

• Otherwise, the conversion fails.

6. If \( T \) is one of the primitive types (byte, char, short, int, long, float, double, boolean) then treat \( T \) as the corresponding wrapper class.

7. If \( T \) extends class Number and \( S \) extends also class Number then convert the source to a number of type \( T \). If the target type cannot hold the value then the conversion fails. However, precision may be lost if a double or float is converted to one of the integer types.

8. If source type \( S \) is not class String, then the conversion fails.

9. The conversion is attempted based on the target type \( T \) from the string \( s \). The following target types are supported:
   - boolean or Boolean - Construct the appropriate boolean type while accepting the following additional values for true and false respectively:
     - yes, no
     - on, off
   - Character - The string \( s \) must have a length of 1, this single character is then converted to a Character object.
   - Locale - The string \( s \) is converted to a Locale using the following syntax (no spaces are allowed between terms).

\[
\text{locale} ::= <\text{java language-code}> ( '_' \text{country})+ \\
\text{country} ::= <\text{java country-code}> ( '_' <\text{java variant-code}>)+
\]

   - Pattern - Create the Pattern object with Pattern.compile(String).
   - Properties - Create a new Properties object and load the properties from the string. The string must follow the format described with the Properties.load method.
   - Enum subclass - Convert the string \( s \) to the appropriate member of the given enum with the Enum.valueOf method. If the string is not one of the enum values, then the conversion must fail.
   - Class - The string \( s \) must conform to the syntax in Syntax for Java types on page 256. This type must be loaded through the Bundle's loadClass method. The resulting class must match any generic constraints on \( T \). If this fails, the conversion fails.

10. If target type \( T \) has a constructor (String), then use this constructor to create an instance with the source string \( s \). This convention caters for many of the built-in Java types such as BigDecimal, BigInteger, File, URL, and so on, as well as for custom types.

If none of the above steps has found a proper conversion than the conversion fails. Failing a conversion must end with throwing an Illegal Argument Exception.

### 121.9.4 Type Converters

A type converter converts a source type to a target type. The source type for a type converter is not constrained. A type converter must support the following methods:

- **canConvert(Object,ReifiedType)** - A light weight method that inspects the object and returns true if it can convert it to the given Reified Type, false otherwise. Converters normally can convert a type \( S \) to a type \( T<...> \). However, converters can convert to multiple types and the value of the source object can influence the returned type. For example, a converter could convert a string to a type based on its content.

- **convert(Object,ReifiedType)** - The actual conversion method. This method should not fail if the canConvert method has returned true.
The `ReifiedType` class provides access to the target class. In a Java 1.4 environment, the `ReifiedType` object will provide a `Class` object for conversion and no type arguments. In a Java 5 environment, the `ReifiedType` object provides access to the reified class as well as the type arguments. Generics and reified types are described in `Generics` on page 291.

Type converters are normal managers with some limitations due to the dependency handling. If they depend on general managers or services then there is a change that cyclic dependencies are created.

Converters must be defined in the `type-converters` element, see `<type-converters>` on page 258, to be registered as a converter. Component instances of managers in this section must implement the `Converter` interface. Converters must also only transitively depend on built-in converters. It must be possible to initialize all converters before any of them are used. Type converters should not use the type conversion before all type converters are fully configured.

Converters are ordered within one definition resource but there is no resource ordering, so the overall ordering is not defined, making it a good practice to concentrate all converters in a single XML definition. The definition ordering is used during type conversion. That is, converters are not ordered by their specialization, a converter that is earlier can convert a more general type will override a converter that is later in the list but could have converted to a more specific type.

Converters must always use the type arguments of the given `ReifiedType`, even if they are running on Java 1.4. The default behavior of the `ReifiedType` will automatically work.

The following example demonstrates how a converter can use generics to use an `AtomicReference<T>` whenever type `T` is supported. Such a type could be for a property like:

```java
class AtomicInteger {
    public void setInteger( AtomicReference<Integer> atomic );
}
```

The Atomic Converter uses the generic argument to convert a source object to an Integer and then creates an `AtomicReference` with this converted object. The definition of the type converter looks like:

```xml
<type-converters>
    <bean class="AtomicConverter">
        <argument ref="blueprintConverter"/>
    </bean>
</type-converters>
```

The Blueprint converter is injected in the constructor of the `AtomicInteger` class, in order to allow the conversion of the generic arguments. The Blueprint built-in type converter must not be used before all type converters are registered because a needed type converter might not have been registered yet. This is the reason type converters should not require type conversion in their initialization because the state of this converter is not well defined at this time.

The conversion class looks like:

```java
public class AtomicConverter {
    Converter bpc;
    public AtomicConverter(Converter bpc) { this.bpc=bpc; }

    public boolean canConvert(Object s, ReifiedType T) {
        return T.getRawClass() == AtomicReference.class && bpc.canConvert(s, T.getActualTypeArgument(0));
    }

    public Object convert(Object s, ReifiedType T) throws Exception {
        Object obj = bpc.convert(
```
s,T.getActualTypeArgument(0);)

    return new AtomicReference<Object>(obj);
  }
}

Any injection that now targets an AtomicReference<T> value will automatically be converted into an AtomicReference of the appropriate type because of the example converter. The following definitions test this behavior:

public class Foo<T extends Integer> {
  public Foo( AtomicReference<T> v) {}
}

<bean id="foo" class="Foo"> <argument value="6"/> </bean>

This definition will create a foo object with the Foo(AtomicReference<T>) constructor. The source type is a string and there is no assignability for an Atomic Reference, so the registered type converters are consulted. The Atomic Converter recognizes that the target T is an AtomicReference class and indicates it can convert. The convert method then uses the generic argument information, which is an Integer object in the example, to convert the string "6" to an Integer object and return the appropriate AtomicReference object.

121.9.5 Built-in Converter

A Blueprint Container must contain an environment manager called blueprintConverter. The related component instance must implement the Converter interface.

The built-in Converter provides access to the provided type converters as well as the built in types. This service provides the type conversion as defined in Type Conversion on page 287.

Injecting a reference to the blueprintConverter environment manager into a bean provides access to all the type conversions that the Blueprint Container and registered type converters are able to perform. However, if this converter is injected in a type converter, then by definition, not all custom type converters are yet registered with the built-in converter. Type converters should therefore in general not rely on type conversion during their construction.

121.9.6 Concrete Types for Interfaces

The Blueprint extender can choose an implementation class when it provides an instance during conversion to an interface as well as when it natively provides an object. The actual implementation class can make a noticeable difference in disambiguation, type conversion, and general behavior. Therefore this sections describe the concrete types an implementation must use for specific interfaces if the platform allows this.

Table 121.18 Implementation types for interfaces

<table>
<thead>
<tr>
<th>Interface/Abstract class</th>
<th>Implementation class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>ArrayList</td>
</tr>
<tr>
<td>List</td>
<td>ArrayList</td>
</tr>
<tr>
<td>Queue</td>
<td>LinkedList</td>
</tr>
<tr>
<td>Set</td>
<td>LinkedListHashSet</td>
</tr>
<tr>
<td>SortedSet</td>
<td>TreeSet</td>
</tr>
<tr>
<td>Map</td>
<td>LinkedHashSet</td>
</tr>
<tr>
<td>SortedMap</td>
<td>TreeMap</td>
</tr>
<tr>
<td>Java 5</td>
<td>LinkedHashMap</td>
</tr>
<tr>
<td>ConcurrentHashMap</td>
<td>ConcurrentHashMap</td>
</tr>
<tr>
<td>ConcurrentMap</td>
<td></td>
</tr>
</tbody>
</table>
If possible, the instances of these types must preserve the definition ordering.

### 121.9.7 Generics

Java 5 introduced the concept of *generics*. Before Java 5, a *type*, was simply a class or interface, both represented by the `Class` object. Generics augment these classes and interfaces with additional *type constraints*. These type constraints are not available on an instance because an instance always references a raw `Class`. For an instance all generic type constraints are *erased*. That is, a `List<Integer>` object is indistinguishable from a `List<String>` object, which are indistinguishable from a `List` object. Objects always refer to a raw `Class` object, this is the one returned from the `getClass` method. This `Class` object is shared between all instances and can therefore not have the actual type constraints (like `String`, `Integer` in the list examples).

When a class is used the compiler captures the type constraints and associates them with the specific use and encodes them in a `Type` object. For example, a field declaration captures the full generic type information:

```java
List<String> strings;
```

A field has a `getGenericArgument` method that provides access to a `Type` object, which is a super interface for all type information in the Java 5 and later runtime. In the previous example, this would be a Parameterized Type that has a raw class of `List` and a type argument that is the `String` class. These constraints are reflectively available for:

- A superclass
- Implemented interfaces
- Fields
- For each method or constructor:
  - Return type
  - Exception types
  - Parameter types

Generics influence the type conversion rules because most of the time the Blueprint extender knows the actual `Type` object for an injection. Therefore, conversion must take place to a type like `T<P1..Pn>`, where `T` is a raw `Class` object and `P1..Pn` form the available type parameters. For a non-parametrized class and for other VMs than 1.4, `n` is always zero, that is no type arguments are available. The `P` arguments are in itself instances of `Type`. The form `T<P1..Pn>` is called the *reified* form. It can be constructed by traversing the `Type` graph and calculating a class that matches the constraints. For example `< extends List<T>>` defines a *wild card* constraint, that has a `List<T>` as reified type, where `T` is a Type Variable defined elsewhere that can have additional constraints. The resulting type must be an instance of `List<T>`. A reified type will use an object implementing `List` for such an example because that is the only class that is guaranteed to be compatible. The rules to reify the different `Type` interfaces are:

- **Class** - A Class represents unparameterized raw type and is reified into `T<>`. For example:
  ```java
  String string;
  ```

- **ParameterizedType** - A Parameterized Type defines a raw type and `1..n` typed parameters. The raw type of the Parameterized Type is also reified and represents `T`. The arguments map directly to the arguments of the reified form. An example of a Parameterized Type is:
  ```java
  Map<String, Object> map;
  ```
• TypeVariable - Represents a Type Variable. A type variable is listed in a generics type declaration, for example in Map<K,V>, the K and V are the type variables. A type variable is bounded by a number of types because it is possible to declare a bounded type like: <A extends Readable&Closeable>. A Type Variable is reified by taking its first bound in reified form, this is the same as in Java 5 where the first bounds is the erasure type. However, this can fail if multiple bounds are present. An example of a Type Variable is:

```java
public <T extends ServiceTracker> void setMap(T st) {}
```

In this example, the parameter st will have a reified type of ServiceTracker.

• WildcardType - A Wildcard Type constrains a type to a set of lower bounds and a set of upper bounds, at least in the reflective API. In the Java 5 and later syntax a Wildcard Type can only specify 0 or one lower and one upper bound, for example <T extends Number> constraints the Type Variable T to at least extend the Number class. A Wildcard Type is reified into its reified upper bound when no lower bound is set, and otherwise it is reified into its reified lower bound. An example of a Wildcard Type is seen in the example of a Type Variable.

• GenericArrayType - A Generic Array Type represents an array. Its component type is reified and then converted to an array. The Reified Type will have the array class as reified class and the type arguments reflect the type arguments of the component type. For example:

```java
public void setLists(List<String>[] lists) {}
```

This example will have a Reified Type of List[].

This specification is written to allow Java 1.4 implementations and clients, the API therefore has no generics. Therefore, the Type class in Java 5 and later cannot be used in the API. However, even if it could use the Type class, using the type classes to create the reified form is non-trivial and error prone. The API therefore provides a concrete class that gives convenient access to the reified form without requiring the usage of the Type class.

The ReifiedType class provides access to the reified form of Class, which is itself and has no type arguments. However, Blueprint extender implementations that recognize Java 5 generics should subclass the ReifiedType class and use this in the conversion process. The subclass can calculate the reified form of any Type subclasses.

121.10 Service Dynamics

The Blueprint Container specification handles the complexities of the dynamic nature of OSGi by hiding the dynamic behavior of the OSGi service registry, at least temporarily. This dynamic behavior is caused by service references that select one or more services that can come and go at runtime.

The Blueprint Container must handle the dynamics in the following way:

• Proxied references - Service reference managers must provide a proxy implementing the specified interfaces, instead of the actual service object it refers to. The proxy must fetch the real service lazily. For reference managers, when a proxy is used, and no candidate is available, a candidate must be found within a limited time. If no candidate service is available during this time, a Service Unavailable Exception must be thrown. The reference-list manager also maintains proxies but these proxies must throw a Service Unavailable Exception immediately when the proxy is used and the backing service is no longer available.

When proxied references are used with stateful services, then the application code must register a reference listener to perform the necessary initialization and cleanup when a new backing service is bound.

• Conditional Service Registrations - The service manager is responsible for registering a service with the OSGi service registry. A service manager is statically dependent on the transitive set of man-
agers that it depends on. If these static dependencies contain mandatory service references, then the manager's service must not be registered when any of these mandatory service references is unsatisfied, see Enabled on page 272.

121.10.1 Damping

When an operation is invoked on an unsatisfied proxy from a reference manager (either optional or mandatory), the invocation must block until either the reference becomes satisfied or a time-out expires (whichever comes first). During this wait, a WAITING event must be broadcast, see Events on page 295.

The default timeout for service invocations is 5 minutes. The optional timeout of the reference element specifies an alternate timeout (in milliseconds). If no matching service becomes available within the timeout, then a Service Unavailable Exception must be thrown. A timeout of zero means infinite and a negative timeout is an error.

For example:

```xml
<reference id="logService"
  interface="org.osgi.service.log.LogService"
  timeout="100000" />

<bean id="bar" class="BarImpl">
  <property name="log" ref="logService"/>
</bean>
```

When this Blueprint Container is instantiated, the reference manager provides a proxy for the Log Service, which gets injected in the log property. If no Log Service is available, then the proxy will have no backing service. If the bar object attempts to log, it will block and if the timeout expires the proxy must throw a Service Unavailable Exception.

If at some later point in time, a Log Service is registered then it becomes satisfied again. If bar now logs a message, the proxy will get the service object again and forward the method invocation to the actual Log Service implementation.

The damping ensures that a mandatory service reference that becomes unsatisfied does not cause the Blueprint Container to be destroyed. Temporary absences of mandatory services are tolerated to allow for administrative operations and continuous operation of as much of the system as possible.

A reference-list manager does not provide damping. It only removes the service proxy from the collection if its service goes away. Using a collection reference manager will never block, it will just have no members if its selection is empty. A timeout attribute is therefore not supported by the reference-list elements. However, the elements are proxied and it is possible that they throw a Service Unavailable Exception when used and the backing service has disappeared. The exceptions for a reference-list proxy will be thrown immediately when the proxy is used.

121.10.2 Iteration

The provided object of a reference-list manager implements the List interface. Depending on the memberType or the optional generics information, it provides a collection that contains the member objects, that is, either proxies to the service object, or ServiceReference objects. These collections are read-only for the receiver, however, their contents can dynamically change due to changes in the selection. The access to these collections with iterators must give a number of guarantees:

- **Safe** - All iterators of reference-list managers must be safe to traverse according to the Iterator interface contract, even while the underlying collection is being modified locally or in another thread. If the hasNext method returns true, the iterator must return a member object on the subsequent next method invocation. If there is no longer a service object available when requested, then a dummy proxy must be returned that throws a Service Unavailable Exception whenever it is used.
• **Visibility** - All the changes made to the collection that affect member objects not yet returned by the iterator must be visible in the iteration. Proxies for new services must be added at the end of the List. Proxies already returned can be affected by changes in the service registry after the iterator has returned them.

After the iterator has returned false for the hasNext method, no more objects can be obtained from it. A List Iterator must not be supported.

### 121.10.3 Mandatory Dependencies

A service manager can have mandatory service reference managers in its transitive dependencies. Such a service manager must ensure that the service object is registered with the OSGi service registry during the runtime phase when all its mandatory service references that it depends on are satisfied. This is called tracking the dependency. A service manager is enabled when all its mandatory references in its dependencies are satisfied.

This tracking only works for dependencies declared directly in the definitions; dependencies established during runtime by calling the getComponentInstance method are not tracked.

In the following example, service manager S has a transitive dependency on the mandatory reference manager M, which means the Blueprint Container must ensure that the service object provided by bean A is registered when reference manager M is satisfied.

```xml
<service id="S" ref="A" interface="com.acme.Foo"/>
<bean id="A" class="com.acme.FooImpl">
   <property name="bar" ref="m"/>
</bean>
<reference id="M" interface="com.acme.Bar" availability="mandatory"/>
```

However, if the dependency from manager A on manager M is not declared but created through code that manipulates the Blueprint Container then the dependency is not tracked.

### 121.11 Blueprint Container

The Blueprint Container has a registry where all top-level managers, as well as environment managers, are registered by their component id. The Blueprint Container can be injected in application code with the environment blueprintContainer manager. For example:

```xml
<bean class="com.acme.FooImpl">
   <property name="container" ref="blueprintContainer"/>
</bean>
```

The Blueprint Container allows application code to get objects that are provided by the top-level managers through the getComponentInstance method. However, the Blueprint Container should not be required to get a component instance; the proper way to use Blueprint is to inject them. This declarative approach makes the Blueprint Container aware of any dependencies; one of the primary goals of a dependency injection framework. The Blueprint Container's introspective features are commonly used for management and other non-application purposes.

The Blueprint Container is registered as a service during the runtime phase so that other bundles can use it for these, and other, purposes.

### 121.11.1 Environment Managers

The Blueprint Container provides a number of environment managers. These managers have defined names and provide convenient access to information about the environment. Environment man-
agers cannot be overridden by explicitly defined managers because it is invalid to define a manager with an existing component id. All component ids starting with blueprint are reserved for this specification and future incarnations.

There is no XML definition for environment managers but their Metadata must be provided as ComponentMetadata objects.

The following ids are used for the environment managers:

- **blueprintContainer** - The Blueprint Container.
- **blueprintBundle** - A manager that provides the Blueprint bundle's Bundle object.
- **blueprintBundleContext** - A manager that provides the Blueprint bundle's BundleContext object.
- **blueprintConverter** - A manager that provides an object implementing the Converter interface.

This represents the built-in conversion facility that the Blueprint Container uses to convert objects. See *Built-in Converter* on page 290.

### 121.11.2 Component Instances

The Blueprint Container provides access to the component instances that the top level managers can provide, as well as their Metadata. The Blueprint Container has the following methods for requesting a component instance and to find out what managers are available:

- **getComponentInstance(String)** - This method will provide a component instance from the component id. If the manager has not been activated yet, it must atomically activate and ensure its explicit and implicit dependencies are activated transitively.
- **getComponentIds()** - Returns a set of component ids in this Blueprint Container. These ids must consist of all top level managers (including calculated ids) and environment managers.

### 121.11.3 Access to Component Metadata

Each of the manager types has specific Component Metadata subtypes associated with it, except Environment managers that use Component Metadata. The Blueprint Container provides access by component id to the Component Metadata of the top level managers. However, managers can also be defined inline, in which case they do not have a component id. Therefore, the Blueprint Container can also enumerate all the managers that are represented by a Metadata sub-interface.

- **getComponentMetadata(String)** - Answer the Component Metadata sub-type for the given component id. Environment managers will return a ComponentMetadata object, the other managers each have their own specific Metadata type.
- **getMetadata(Class)** - Answer a collection with the Metadata of the given type, regardless if it is defined as/in a top-level or inlined manager. For example, `getMetadata(ServiceMetadata.class)` returns all Service Metadata in the Blueprint container. This includes all top level managers as well as any inlined managers. For Environment Managers, this method returns a ComponentMetadata object.

### 121.11.4 Concurrency

A Blueprint Container must be thread safe. Each method must handle the case when multiple threads access the underlying registry of managers. Activation of managers must be atomic. That is, other threads must be blocked until a manager is completely activated.

The Blueprint Container must handle reentrant calls.

### 121.12 Events

The Blueprint Container must track all Blueprint Listener services and keep these listeners updated of the progress or failure of all its managed bundles. The Blueprint Listener is kept informed by
sending it events synchronously. These events are therefore normally delivered in order but in ex-
ceptional cases this can be seen out of order for a listener when new events are initiated synchro-
nously from within a callback. Therefore, Blueprint Listener services should see the event as a noti-
cication, where actual work should be processed on another thread.

Blueprint Events must be sent to each registered Blueprint Listener service. This service has the fol-
lowing method:

- `blueprintEvent(BlueprintEvent)` - Notify the listener of a new Blueprint Event. These events are
  send synchronously with their cause. That is, all listeners must be notified before the Blueprint
  Container continues to the next step.

The events must be delivered as `BlueprintEvent` objects. The event types that they represent, and the
data that these objects carry, is further described in `Blueprint Event` on page 296.

A Blueprint Listener services must be given the initial state of all managed bundles before normal
processing starts, see `Replay` on page 296.

Blueprint Listener services that throw Exceptions or do not return in a reasonable time as judged by
the Blueprint extender implementation, should be logged, if possible, and further ignored.

### 121.12.1 Blueprint Event

The Blueprint Event supports the following event types:

- **CREATING** - The Blueprint extender has started creating a Blueprint Container for the bundle.
- **GRACE_PERIOD** - The Blueprint Container enters the grace period. This event can be repeated
  multiple times when the list of dependencies changes due to changes in the service registry.
- **CREATED** - The Blueprint Container is ready. The application is now running.
- **WAITING** - A service reference is blocking because of unsatisfied mandatory dependencies. This
  event can happen multiple times in a row.
- **DESTROYING** - The Blueprint Container is being destroyed because the Blueprint bundle or Blue-
  print extender has stopped.
- **DESTROYED** - The Blueprint Container is completely destroyed.
- **FAILURE** - An error occurred during the creation of the Blueprint Container.

The Blueprint Event provides the following methods:

- `getBundle()` - The Blueprint bundle
- `getCause()` - Any occurred expection or null
- `getDependencies()` - A list of filters that specify the unsatisfied mandatory references.
- `getExtenderBundle()` - The Blueprint extender bundle.
- `getTimestamp()` - The time the event occurred
- `getType()` - The type of the event.
- `isReplay()` - Indicates if the event is a replay (true) or if it is a new event (false), see `Replay` on page
  296.

### 121.12.2 Replay

The Blueprint Extender must remember the last Blueprint Event for each ready bundle that it man-
ages, see `Initialization Steps` on page 250. During the (synchronous) service registration event of a
Blueprint Listener service, the Blueprint extender must inform the Blueprint Listener service about
all its managed bundles by sending it the last known event for each bundle the Blueprint extender
manages. This initial event is called the `replay` event, and is marked as such.

The replay event must be delivered to the Blueprint Listener service as the first event, before any
other event is delivered, during the registration of the Blueprint Listener service. That is, the blue-
printEvent method must have returned before the first non-replay event can be delivered and no events must be lost. The replay events must be sent every time a Blueprint Listener service is registered.

The set of managed bundles is defined by bundles that are active and are managed by the Blueprint extender, even if their initialization ended in failure.

The BlueprintEvent object for a replay event must return true for the isReplay() method in this situation, and false in all other situations.

### 121.12.3 Event Admin Mapping

When the Event Admin service is present, the Blueprint extender must create an Event Admin event for each defined Blueprint Event. This Event Admin event must be asynchronously given to the Event Admin service with the postEvent method.

The topic of the Event Admin event is derived from the Blueprint event type with a fixed prefix. All topics must have the prefix of:

```
TOPIC_BLUEPRINT_EVENTS
```

After this prefix, the name of the Blueprint Event type must be used as the suffix. That is, CREATING, GRACE_PERIOD, etc. For example, org/osgi/service/blueprint/container/GRACE_PERIOD.

For each Blueprint event the following properties must be included:

- **TYPE** - The type of the Event, see Blueprint Event on page 296.
- **BUNDLE** - (Bundle) The Bundle object of the Blueprint bundle.
- **BUNDLE_ID** - (Long) The id of the Blueprint bundle.
- **BUNDLE_SYMBOLICNAME** - (String) The Bundle Symbolic Name of the Blueprint bundle.
- **BUNDLE_VERSION** - (Version) The version of the Blueprint bundle.
- **EXTENDER_BUNDLE** - (Bundle) the Bundle object of the Blueprint extender bundle.
- **EXTENDER_BUNDLE_ID** - (Long) The id of the Blueprint extender bundle.
- **EXTENDER_BUNDLE_SYMBOLICNAME** - (String) The Bundle Symbolic Name of the Blueprint extender bundle.
- **EXTENDER_BUNDLE_VERSION** - (Version) The version of the Blueprint extender bundle.
- **TIMESTAMP** - (Long) The time when the event occurred.
- **CAUSE** - (Throwable) The failure cause, only included for a FAILURE event.
- **DEPENDENCIES** - (String[]) The filter of an unsatisfied service reference. Can only appear in a GRACE_PERIOD, WAITING or FAILURE event caused by a time-out.
- **EVENT** - (BlueprintEvent) The BlueprintEvent object that caused this event.

The property names for Blueprint Listener events may be conveniently referenced using the constants defined in the org.osgi.service.event.EventConstants and EventConstants interfaces.

The Event Admin events do not follow the replay model in use for Blueprint Listener services. That is, the Event Admin must only be kept informed about events as they occur.

### 121.13 Class Loading

The module layer in OSGi provides advanced class loading rules that potentially can cause bundles to live in different class spaces. This means that not all bundles can collaborate because the classes involved in the collaboration can come from different class loaders, which results in confusing Class Cast Exceptions on classes with the same name. It is therefore crucial that the Blueprint Container uses the Bundle Context and the bundle class loader of the Blueprint bundle for all actions that are made on behalf of the Blueprint bundle. Especially, access to the OSGi service registry must
use the Bundle Context of the Blueprint bundle. Any dynamic class loading must use the Blueprint bundle's loadClass method. The normal OSGi mechanics will then ensure class space consistency for resolved bundles.

121.13.1 Blueprint Extender and Bundle Compatibility

For many Blueprint bundles, there is no class space compatibility issue. These bundles do not use any Blueprint classes and are therefore by definition compatible with any extender. However, if the Blueprint bundle uses some of the Blueprint packages, it must import these packages. Blueprint Containers must verify that they are type compatible with the Blueprint bundle before they attempt to manage it. See Type Compatibility on page 298.

121.13.2 XML and Class Loading

The Blueprint definition resources contain textual references to classes. These textual references will be loaded with the class loader of the Blueprint bundle. This implies that all the classes of provided component instances must be either imported or available from the bundle.

The Blueprint specification has the following attributes and elements that can cause imports:

- class
- value-type
- interface
- interfaces
- type
- key-type

All these attributes and elements are defined with the Tclass and Ttype XML Schema type for the Blueprint namespace. The Tclass defines simple class names, and Ttype defines types defined in Syntax for Java types on page 256.

121.13.3 Foreign Bundle Context

When using the Blueprint Container in its Blueprint bundle, the types that the managers provide are guaranteed to be compatible with the caller.

When using a Blueprint Container service in another bundle (for example, getting it as a service) then there is no guarantee of type compatibility or even visibility between the versions of the types of the returned managers, and the versions of the types visible to the caller. Care must therefore be taken when casting the return value of the getComponentInstance method to a more specific type.

121.13.4 Converters and Class Loading

A converter is closely coupled to its target class. If the converter comes from another bundle, then the converter bundle must ensure class space consistency between the converter implementation and the target class. This can be achieved by specifying the target class in the uses directive.

For example:

```
Export-Package:
    com.converters.ac;uses="com.converters.dc"
```

A bundle that references a type converter defined in the Blueprint bundle does not need to export that type. When creating a Blueprint Container, the extender bundle uses the class loader of the Blueprint bundle.

121.13.5 Type Compatibility

Two bundles are type compatible for a given class if they both load the same class object, or if either bundle cannot load the given class.
To mitigate type incompatibility problems, a Blueprint extender must export the org.osgi.service.blueprint package. In the uses: directive, it should list any packages of classes that can be shared between the Blueprint extender and the Blueprint bundle. Blueprint bundles should import this package.

### 121.13.6 Visibility and Accessibility

The Blueprint Container must load any classes it needs through the Blueprint bundle's loadClass method. If a class can not be loaded, then the initialization fails. Class loading issues are further discussed in [Class Loading](#) on page 297.

The Blueprint Container must respect the accessibility of the class and any of its members. That is, the Blueprint Container must not use the setAccessibility method. All classes and reflected members must therefore be declared public or be implicitly public like the default constructor.

### 121.14 Metadata

An important aspect of the Blueprint specification is the so called metadata interfaces. These interfaces are used in the Blueprint Container to enable programmatic access to the XML definitions. During the parsing phase the Blueprint Container reads the XML and converts it to an object implementing the appropriate interface.

The XML elements and XML Schema types map to the Metadata interfaces. For example, `<bean>` maps to BeanMetadata. However, in several cases, the attributes and/or sub-elements in the Metadata interfaces are merged when possible. For example, the interface attribute and interfaces element in the service element are merged in the ServiceMetadata class' getInterfaces() method.

The interfaces are arranged in a comprehensive hierarchy that reflects their usage and constraints. This hierarchy is depicted in Figure 121.12 on page 300.

The hierarchy can roughly be divided in two parts. The first part is the sub-interfaces of the ComponentMetadata interface. These interfaces are defining the configuration data of the top-level and inlined managers. The manager's component instance(s) are injected with values during runtime. The configuration of how to create a specific value is also described with Metadata interfaces. For example, a Map object is described with configuration information in the MapMetadata interface. The hierarchy makes it clear that Component Metadata is also a value that can be injected. Keys in maps or properties can not be null. This is the reason the hierarchy is split at the top into a null value branch and a branch that can only generates non-null values.

The Target interface describes managers that can be used as the target for the reference listener or the registration listener, or a ref.
Figure 121.12 Metadata Interfaces Hierarchy

121.15 Blueprint XML Schema

The Blueprint schema included in this specification can be found in digital form at [9] OSGi XML Schemas.

```xml
<xsd:schema xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  elementFormDefault="qualified" attributeFormDefault="unqualified"
  version="1.0.1">

  <!-- Schema elements for core component declarations -->

  <xsd:complexType name="Tcomponent" abstract="true">
    <xsd:annotation>
      <xsd:documentation>
        <![CDATA[
This is the XML Schema for the OSGi Blueprint service 1.0.0 development descriptor. Blueprint configuration files using this schema must indicate the schema using the blueprint/v1.0.0 namespace. For example, <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

if used as a qualified namespace, "bp" is the recommended namespace prefix.
]]>
      </xsd:documentation>
    </xsd:annotation>

    <!-- Schema elements for core component declarations -->

    </xsd:complexType>
</xsd:schema>
```

The Tcomponent type is the base type for top-level Blueprint components. The <bean>, <reference>, <service>, and <reference-list> elements are all derived from the Tcomponent type. This type defines an id attribute that is used create references between different components. Component elements can also be inlined within other component definitions. The id attribute is not valid when inlined.
<xsd:attribute name="id" type="xsd:ID" />
<xsd:attribute name="activation" type="Tactivation">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
The activation attribute for this component. This can either be "eager" or "lazy". If not specified, it defaults to default-activation attribute of the enclosing <blueprint> element.
]]></xsd:documentation>
  </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="depends-on" type="TdependsOn">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
depends-on identifies (by id) other components that this component depends on. The component only be activated after the depends-on components are successfully activated. Also, if there are <reference> or <reference-list> elements with unsatisfied mandatory references, then the depends-on relationship will also be used to determine whether this service is enabled or not.
]]></xsd:documentation>
  </xsd:annotation>
</xsd:attribute>
</xsd:complexType>
<xsd:element name="blueprint" type="Tblueprint">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
The <blueprint> element is the root element for a blueprint configuration file. A blueprint configuration has two sections. The first section (contained within the <type-converters> element) identifies components that are used for converting values into different target types. The type converters are optional, so the file does not need to specify a type converter section. Following the type converters are the component definitions. Components are <bean>, <service>, <reference>, and <reference-list> elements that identify the bundle components that will be managed by the blueprint service.
]]></xsd:documentation>
  </xsd:annotation>
</xsd:element>
</xsd:complexType>
<xsd:element name="description" type="Tdescription" minOccurs="0" />
<xsd:element name="type-converters" type="Ttype-converters" minOccurs="0" maxOccurs="1" />
<!-- top-level components -->
<xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:element name="service" type="Tservice" />
  <xsd:element name="reference-list" type="Treference-list" />
  <xsd:element name="bean" type="Tbean" />
  <xsd:element name="reference" type="Treference" />
  <xsd:any namespace="##other" processContents="lax" />
</xsd:choice>
</xsd:sequence>
</xsd:element>
<!-- Defaults-->
<xsd:documentation><![CDATA[
Specifies the default activation setting that will be defined for components. If not specified, the global default is "eager". Individual components may override the default value.
]]>
</xsd:documentation>
</xsd:annotation>
</xsd:attribute>
<xsd:attribute name="default-timeout" type="Timeout" default="300000">
<xsd:annotation>
<xsd:documentation><![CDATA[
Specifies the default timeout value to be used when operations are invoked on unassigned service references. If the reference does not change to a satisfied state within the timeout window, an error is raised on the method invocation. The default timeout value is 300000 milliseconds and individual <reference> element can override the specified configuration default.
]]>
</xsd:documentation>
</xsd:annotation>
</xsd:attribute>
<xsd:attribute name="default-availability" type="Tavailability" default="mandatory">
<xsd:annotation>
<xsd:documentation><![CDATA[
Specifies the default availability value to be used for <reference>, and <reference-list> components. The normal default is "mandatory", and can be changed by individual service reference components.
]]>
</xsd:documentation>
</xsd:annotation>
</xsd:attribute>
<xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
<xsd:complexType name="Ttype-converters">
<xsd:annotation>
<xsd:documentation><![CDATA[
The type used for the <type-converters> element. The <type-converters> section is a set of <bean>, <ref>, or <reference> elements that identify the type converter components.
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:choice minOccurs="0" maxOccurs="unbounded">
<xsd:element name="bean" type="Tbean" />
<xsd:element name="reference" type="Treference" />
<xsd:element name="ref" type="Tref" />
<xsd:any namespace="##other" processContents="lax" />
</xsd:choice>  
</xsd:complexType>
</xsd:group name="GtargetComponent">
<xsd:annotation>
<xsd:documentation><![CDATA[
Components that provide a reasonable target for injection used for listeners, etc.
-->
</xsd:documentation>
</xsd:annotation>
</xsd:group>
<xsd:choice>
  <xsd:element name="bean" type="Tinlined-bean" />
  <xsd:element name="reference" type="Tinlined-reference" />
  <xsd:element name="ref" type="Tref" />
  <xsd:any namespace="##other" processContents="lax"/>
</xsd:choice>
</xsd:group>

<xsd:group name="GAllComponents">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      An all components is used in contexts where all component element types are values. The set of component elements contains <bean>, <service>, <reference>, <reference-list> and <ref>.]]>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice>
    <xsd:element name="service" type="Tinlined-service" />
    <xsd:element name="reference-list" type="Tinlined-reference-list"/>
    <xsd:group ref="GtargetComponent"/>
  </xsd:choice>
</xsd:group>

<xsd:group name="GbeanElements">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      A bean elements is a reusable definition of the elements allowed on a <bean> element.]]>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="description" type="Tdescription" minOccurs="0"/>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="argument" type="Targument"/>
      <xsd:element name="property" type="Tproperty"/>
      <xsd:any namespace="##other" processContents="lax"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:group>

<xsd:complexType name="Tbean">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      The type definition for a <bean> component. The <bean> attributes provide the characteristics for how to create a bean instance. Constructor arguments and injected properties are specified via child <argument> and <property> elements.]]>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="Tcomponent">
      <xsd:group ref="GbeanElements"/>
      <xsd:attribute name="class" type="Tclass"/>
      <xsd:attribute name="init-method" type="Tmethod"/>
      <xsd:attribute name="destroy-method" type="Tmethod"/>
      <xsd:attribute name="factory-method" type="Tmethod"/>
      <xsd:attribute name="factory-ref" type="Tidref"/>
      <xsd:attribute name="scope" type="Tscope"/>
      <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
The Tinlined-bean type is used for inlined (i.e. non top level) <bean> elements. Those elements have some restrictions on the attributes that can be used to define them.

An argument used to create an object defined by a <bean> component. The <argument> elements are the arguments for the bean class constructor or passed to the bean factory method.

The type, if specified, is used to disambiguate the constructor or method signature. Arguments may also be matched up with arguments by explicitly specifying the index position. If the index is used, then all <argument> elements for the bean must also specify the index.

The value and ref attributes are convenience shortcuts to make the <argument> tag easier to code. A fuller set of injected values and types can be specified using one of the "value" type elements.
<xsd:complexType name="Tkey">
  <xsd:annotation>
    <! [CDATA[
      The Tkey type defines the element types that are permitted
      for Map key situations. These can be any of the "value"
      types other than the <null> element.
    ]]>
  </xsd:annotation>
  <xsd:group ref="GnonNullValue" />
</xsd:complexType>

<!-- reference -->
<xsd:complexType name="Treference">
  <xsd:annotation>
    <! [CDATA[
      The Treference type defines the <reference> element. These
      are instances of the TserviceReference type, with the addition
      of a timeout attribute. If the timeout is not specified,
      the default-timeout value is inherited from the encapsulating
      <blueprint> definition.
    ]]>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="TserviceReference">
      <xsd:sequence>
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
          processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="timeout" type="Ttimeout" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Tinlined-reference">
  <xsd:annotation>
    <! [CDATA[
      The Tinlined-reference type is used for inlined (i.e. non top level)
      <reference> elements.
    ]]>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:restriction base="Treference">
      <xsd:sequence>
        <xsd:group ref="GserviceReferenceElements" />
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
          processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="id" use="prohibited" />
      <xsd:attribute name="depends-on" type="TdependsOn" />
      <xsd:attribute name="activation" use="prohibited"
        fixed="lazy" />
      <xsd:attribute name="interface" type="Tclass" />
      <xsd:attribute name="filter" type="xsd:normalizedString" />
      <xsd:attribute name="component-name" type="Tidref" />
      <xsd:attribute name="availability" type="Tavailability" />
      <xsd:attribute name="timeout" type="Ttimeout" />
      <xsd:anyAttribute namespace="##other"
        processContents="lax" />
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
<!-- reference-list -->
<xsd:complexType name="Treference-list">
  <!-- The reference-list builds in the characteristics of the TserviceReference type to define characteristics of the <reference-list>. This adds in the characteristics that only apply to collections of references (e.g., member-type). -->
  <xsd:annotation>
    <xsd:documentation>
      <!-- The reference-list builds in the characteristics of the TserviceReference type to define characteristics of the <reference-list>. This adds in the characteristics that only apply to collections of references (e.g., member-type). -->
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="TserviceReference">
      <xsd:sequence>
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="member-type" type="Tservice-use" default="service-object" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- Tinlined-reference-list -->
<xsd:complexType name="Tinlined-reference-list">
  <xsd:annotation>
    <xsd:documentation>
      <!-- The Tinlined-reference-list type is used for inlined (i.e. non top level) <reference-list> elements. -->
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:restriction base="Treference-list">
      <xsd:sequence>
        <xsd:group ref="GserviceReferenceElements" />
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="id" use="prohibited" />
      <xsd:attribute name="depends-on" type="TdependsOn" />
      <xsd:attribute name="activation" use="prohibited" fixed="lazy" />
      <xsd:attribute name="interface" type="Tclass" />
      <xsd:attribute name="filter" type="xsd:normalizedString" />
      <xsd:attribute name="component-name" type="Tidref" />
      <xsd:attribute name="availability" type="Tavailability" />
      <xsd:attribute name="member-type" type="Tservice-use" default="service-object" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

<!-- Reference base class -->
<xsd:complexType name="TserviceReference">
  <xsd:annotation>
    <xsd:documentation>
      <!-- TserviceReference is the base element type used for <reference> and <reference-list> elements. This type defines all of the characteristics common to both sorts of references. -->
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="Tcomponent">
      <xsd:sequence>
        <xsd:group ref="GserviceReferenceElements" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:sequence>

<xsd:attribute name="interface" type="Tclass">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      The interface that the OSGi service must implement and that will be implemented by the proxy object.
      This attribute is optional.
      ]]]>
    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>

<xsd:attribute name="filter" type="xsd:normalizedString">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      A filter string used to narrow the search for a matching service reference.
      ]]]>
    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>

<xsd:attribute name="component-name" type="Tidref">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      An optional specifier that can be used to match a service definition to one created by a specific blueprint component.
      ]]]>
    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>

<xsd:attribute name="availability" type="Tavailability">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
      Use to control the initial processing of service references at blueprint context startup. "mandatory" indicates the context should not start unless the service is available within the specified context startup period. "optional" indicates availability of this service is not a requirement at bundle startup.
      ]]]>
    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>

<xsd:attribute name="availability" type="Tavailability" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:group name="GserviceReferenceElements">
  <xsd:sequence>
    <xsd:element name="description" type="Tdescription"
      minOccurs="0" />
    <xsd:element name="reference-listener" type="TreferenceListener"
      minOccurs="0" maxOccurs="unbounded">
      <xsd:annotation>
        <xsd:documentation>
          <![CDATA[
          A definition of a listener that will watch for bind/unbind events associated with the service reference. The targetted listener can be a <ref> to a <bean> or <reference> element, or an inline <bean> or <reference>.
          ]]]>
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
</xsd:group>
<xsd:complexType name="TreferenceListener">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
TReferenceListener defines a reference listener that is attached to a <reference> or <reference-list> element. The listener object can be specified as a <ref> or as an inline <bean> or <reference> component. Listener events are mapped to the indicated bind or unbind methods.
]]></xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:group ref="GtargetComponent" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="ref" type="Tidref" />
  <xsd:attribute name="bind-method" type="Tmethod" />
  <xsd:attribute name="unbind-method" type="Tmethod" />
</xsd:complexType>

<xsd:simpleType name="Tactivation">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
Tactivation defines the activation type for components. This is used in this schema by the <blueprint> default-activation attribute and the activation attribute.
]]></xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="eager" />
    <xsd:enumeration value="lazy" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Tavailability">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
Tavailability defines an availability attribute type. This is used in this schema by the <blueprint> default-availability attribute and the <reference> and <reference-list> availability attribute.
]]></xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="mandatory" />
    <xsd:enumeration value="optional" />
  </xsd:restriction>
</xsd:simpleType>

<-- service -->

<xsd:complexType name="Tservice">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
Tservice is the type for services exported by this blueprint bundle. Services are sourced by either a <ref> to a <bean> component or an <inline> bean component.
]]></xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="Tcomponent">
      <xsd:sequence>
        <xsd:group ref="GserviceElements" />
      </xsd:sequence>
      <xsd:attribute name="interface" type="Tclass" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:annotation>
  <xsd:documentation><![CDATA[
The interface that this OSGi service will provide.
]]></xsd:documentation>
</xsd:annotation>
<xsd:attribute name="ref" type="Tidref">
    <xsd:annotation>
        <xsd:documentation>
            <![CDATA[
                The ref attribute can be used to specify the component that provides
                the object exported as an OSGi service.
            ]]>
        </xsd:documentation>
    </xsd:annotation>
</xsd:attribute>

<xsd:attribute name="auto-export" type="TautoExportModes" default="disabled">
    <xsd:annotation>
        <xsd:documentation>
            <![CDATA[
                If set to a value different from "disabled", the Blueprint Container
                will introspect the target to discover the set of interfaces or classes
                that the service will be registered under.
            ]]>
        </xsd:documentation>
    </xsd:annotation>
</xsd:attribute>

<xsd:attribute name="ranking" type="xsd:int" default="0">
    <xsd:annotation>
        <xsd:documentation>
            <![CDATA[
                A service ranking value that is added to the service properties
                the service will be published with.
            ]]>
        </xsd:documentation>
    </xsd:annotation>
</xsd:attribute>

<xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Tinlined-service">
    <xsd:annotation>
        <xsd:documentation>
            <![CDATA[
                The Tinlined-service type is used for inlined (i.e. non top level)
                <service> elements.
            ]]>
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:restriction base="Tservice">
            <xsd:sequence>
                <xsd:element name="description" type="Tdescription" minOccurs="0" />
            </xsd:sequence>
            <xsd:attribute name="id" use="prohibited" />
            <xsd:attribute name="depends-on" type="TdependsOn" />
            <xsd:attribute name="activation" use="prohibited" fixed="lazy" />
            <xsd:attribute name="interface" type="Tclass" />
            <xsd:attribute name="ref" type="Tidref" />
            <xsd:attribute name="auto-export" type="TautoExportModes" default="disabled" />
            <xsd:attribute name="ranking" type="xsd:int" default="0" />
            <xsd:anyAttribute namespace="##other" processContents="lax" />
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="interfaces" type="Tinterfaces"
    minOccurs="0">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
A collection of one or more interface class names this service
will be registered under. The <service> element also has
a shortcut interface attribute for the usual case of just
a single interface being used. This also cannot be used if
the auto-export attribute is used.
]]>
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="service-properties" type="TserviceProperties"
    minOccurs="0">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
The service provided when the service is registered. The service
properties are similar to map elements, but the keys must always
be strings, and the values are required to be in a narrower range.
]]>
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="registration-listener" type="TregistrationListener"
    minOccurs="0" maxOccurs="unbounded">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
A set of 0 or more registration listeners attached to this service
component. The registration listeners will be notified whenever the
service is registered or unregistered from the framework service
registry.
]]>
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:group name="GserviceElements">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
A set of service elements.
]]>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:group ref="GbaseServiceElements" />
    <xsd:group ref="GtargetComponent" minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>
          <![CDATA[
A service definition can use any of the target types as an inline element
as well.
]]>
        </xsd:documentation>
      </xsd:annotation>
    </xsd:group>
  </xsd:sequence>
</xsd:group>

<xsd:complexType name="TregistrationListener">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
A registration listener definition. The target registration listener
can be either a <ref> to a <bean> or <service> component, or an inline
<bean> or <service> component definition. The registration-method and
unregistration-method attributes define the methods that will be called
for the respective events.
]]>
    </xsd:documentation>
  </xsd:annotation>
</xsd:complexType>
For the very common case of using a <ref> to a listener component, the ref attribute may also be used as a shortcut.

]]>
</xsd:documentation>
</xsd:annotation>
</xsd:sequence>
<xsd:attribute name="ref" type="Tidref" />
<xsd:attribute name="registration-method" type="Tmethod" />
<xsd:attribute name="unregistration-method" type="Tmethod" />
</xsd:complexType>

<!-- Values -->

<xsd:group name="Gvalue">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
The set of "value" types that can be used in any place a value can be specified. This set includes the <ref> and <idref> elements, any of the component types (<bean>, <service>, etc.) as inline components, the generic <value> element for types sourced from string values, any of the collection types (<set>, <list>, <array>, <map>, <props>), and the <null> type to inject a null value.
]]></xsd:documentation>
  </xsd:annotation>
  <xsd:choice>
    <xsd:group ref="GnonnullValue" />
    <xsd:element name="null" type="Tnull" />
  </xsd:choice>
</xsd:group>

<xsd:complexType name="Tnull">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
The definition for a <null> value type.
]]></xsd:documentation>
  </xsd:annotation>
</xsd:complexType>

<xsd:group name="GnonnullValue">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
The set of "value" types that can be used in any place a non-null value can be specified. This set includes the <ref> and <idref> elements, any of the component types (<bean>, <service>, etc.) as inline components, the generic <value> element for types sourced from string values, and any of the collection types (<set>, <list>, <array>, <map>, <props>). The <null> type is NOT a member of this group.
]]></xsd:documentation>
  </xsd:annotation>
  <xsd:choice>
    <xsd:group ref="GallComponents" />
    <xsd:element name="idref" type="Tref" />
    <xsd:element name="value" type="Tvalue" />
    <xsd:element name="list" type="Tcollection" />
    <xsd:element name="set" type="Tcollection" />
    <xsd:element name="map" type="Tmap" />
    <xsd:element name="array" type="Tcollection" />
    <xsd:element name="props" type="Tprops" />
  </xsd:choice>
</xsd:group>

<xsd:complexType name="Tref">
  <xsd:annotation>
    <xsd:documentation><![CDATA[

OSGi Enterprise Release 6  Page 311

Licensed under the OSGi Distribution and Feedback License, Version 2.0. Copyright © OSGi Alliance. All Rights Reserved.
Tref is the type used for <ref> elements. This specifies a required component id for the reference component.

Tvalue is the type used for <value> elements. The <value> element is used for types that can be created from a single string value. The string value is the data value for the element. The optional type attribute allows a target conversion value to be explicitly specified.

Tcollection is the base schema type for different ordered collection types. This is shared between the <array>, <list>, and <set> elements.

Tprops is the type used by the <props> value element. The prop elements are pairs of string-valued keys and values.

Tprop is a single property element for a <props> value type. The property value can be specified using either the attribute, or as value data for the property element.
<![CDATA[
Tmap is the base type used for <map> elements. A map may have a default value type specified, so it inherits from the TtypeCollection type. A key type can also be specified, and the map members are created from the entry elements, which require a key/value pair.
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:complexContent>
<xsd:extension base="TtypedCollection">
<xsd:sequence>
<xsd:element name="entry" type="TmapEntry" minOccurs="0" maxOccurs="unbounded" />  
</xsd:sequence>
<xsd:attribute name="key-type" type="Ttype" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<!-- 'entry' element type -->
<xsd:complexType name="TmapEntry">
<xsd:annotation>
<![CDATA[
TmapEntry is used for <entry> elements nested inside of a <map> element. Each <entry> instance defines a key/value pair that will be added to the Map. Both the keys and values may be arbitrary types. Keys must not be <null> but <null> is permitted for entry values. A default type can be specified for both the keys and the values, but individual keys or values can override the default.
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="key" type="Tkey" minOccurs="0" />  
<xsd:group ref="Gvalue" minOccurs="0" />  
</xsd:sequence>
<xsd:attribute name="key" type="TstringValue" />  
<xsd:attribute name="key-ref" type="Tidref" />  
<xsd:attribute name="value" type="TstringValue" />  
<xsd:attribute name="value-ref" type="Tidref" />  
</xsd:complexType>

<!-- 'service property' element type -->
<xsd:complexType name="TserviceProperties">
<xsd:annotation>
<![CDATA[
TserviceProperty is used for <service-properties> elements. The syntax is similar to what is defined for <map>, but keys must be string values and there are no type defaults that can be specified. created from the entry elements, which require a key/value pair.
]]>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="entry" type="TservicePropertyEntry" minOccurs="0" maxOccurs="unbounded" />  
<xsd:element namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />  
</xsd:sequence>
</xsd:complexType>

<!-- 'entry' element type -->
<xsd:complexType name="TservicePropertyEntry">
  <xsd:annotation>
    <xsd:documentation>
      <! [CDATA[
        TservicePropertyEntry is an entry value used for the <service-properties> element. This does not allow a child <key> element and there are no key-ref or value-ref attributes.
      ]]> 
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:group ref="Gvalue" minOccurs="0" /> 
  </xsd:sequence>
  <xsd:attribute name="key" type="TstringValue" use="required" />
  <xsd:attribute name="value" type="TstringValue" />
</xsd:complexType>

<!-- General types -->
<xsd:complexType name="Tdescription" mixed="true">
  <xsd:annotation>
    <xsd:documentation>
      <! [CDATA[
        A generic <description> element type to allow documentation to added to the blueprint configuration.
      ]]> 
    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice minOccurs="0" maxOccurs="unbounded" />
</xsd:complexType>

<xsd:complexType name="Tinterfaces">
  <xsd:annotation>
    <xsd:documentation>
      <! [CDATA[
        The type definition for the <interfaces> element used for <service>
      ]]> 
    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice minOccurs="1" maxOccurs="unbounded">
    <xsd:element name="value" type="TinterfaceValue" />
  </xsd:choice>
</xsd:complexType>

<xsd:simpleType name="TinterfaceValue">
  <xsd:annotation>
    <xsd:documentation>
      <! [CDATA[
        TinterfaceValue is used for subelements of the <interfaces> element. This is just a <value>xxxxx</value> element where the contained value is the name of an interface class.
      ]]> 
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="Tclass" />
</xsd:simpleType>

<xsd:simpleType name="Tclass">
  <xsd:annotation>
    <xsd:documentation>
      <! [CDATA[
        TClass is a base type that should be used for all attributes that refer to java class names.
      ]]> 
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:NCName" />
</xsd:simpleType>

<xsd:simpleType name="Ttype">
  <xsd:annotation>
    <xsd:documentation>
      <! [CDATA[
        Ttype is a base type that refer to java types such as classes or arrays.
      ]]> 
    </xsd:documentation>
  </xsd:annotation>
</xsd:simpleType>
<xsd:simpleType name="Tmethod">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
        Tmethod is a base type that should be used for all attributes that refer to java method names.
      ]]>  
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:NCName" />
</xsd:simpleType>

<!-- Should be used for all attributes and elements that refer to method names -->

<xsd:simpleType name="Tidref">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
        Tidref is a base type that should be used for all attributes that refer to component ids.
      ]]>  
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:NCName" />
</xsd:simpleType>

<xsd:simpleType name="TstringValue">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
        TstringValue is a base type that should be used for all attributes that refer to raw string values
      ]]>  
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:normalizedString" />
</xsd:simpleType>

<xsd:simpleType name="TautoExportModes">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
        TautoExportModes is a base type that should be used for export-mode attributes.
      ]]>  
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="disabled" />
    <xsd:enumeration value="interfaces" />
    <xsd:enumeration value="class-hierarchy" />
    <xsd:enumeration value="all-classes" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Ttimeout">
  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
        Ttimeout is a base type that should be used for all attributes that specify timeout values
      ]]>  
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:unsignedLong" />
</xsd:simpleType>
<xsd:simpleType name="TdependsOn">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
    TdependsOn is a base type that should be used for all attributes that specify depends-on relationships
    ]]>)
  </xsd:documentation>
  <xsd:restriction>
    <xsd:simpleType>
      <xsd:list itemType="Tidref" />
    </xsd:simpleType>
    <xsd:minLength value="1" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Tscope">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="singleton" />  
        <xsd:enumeration value="prototype" />
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:QName">
        <xsd:pattern value=".+:.+"> 
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="Tservice-use">
  <xsd:annotation>
    <xsd:documentation><![CDATA[
    Indicates the type of object that will be placed within the reference collection. "service-object" indicates the collection contains blueprint proxies for imported services. "service-reference" indicates the collection contains ServiceReference objects matching the target service type.
    ]]>)
  </xsd:documentation>
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="service-object" />  
    <xsd:enumeration value="service-reference" />
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
121.16 Security

121.16.1 Blueprint Extender

A Blueprint Extender must use the Bundle Context of the Blueprint bundle. This will ensure that much of the resources allocated will be used on behalf of the Blueprint bundle. However, most Java 2 permissions will also verify the stack and this will inevitably include the Blueprint extender’s code. Therefore, the Blueprint extender will require the combined set of permissions needed by all Blueprint bundles. It is therefore likely that in practical situations the Blueprint extender requires All Permission.

The Blueprint bundle requires permission for all actions that are done by the Blueprint Container on behalf of this bundle. That is, the Blueprint Container must not give any extra permissions to the Blueprint bundle because it is being extended.

A Blueprint Container must therefore use a doPrivileged block around all actions that execute code on behalf of the Blueprint bundle. This doPrivileged block must use an Access Control Context that represents the permissions of the Blueprint bundle.

For example, if a Blueprint bundle defines the following bean:

```xml
<bean class="java.lang.System" factory-method="exit">
  <argument value="1"/>
</bean>
```

Then the Blueprint bundle must have the proper permission to exit the system or the Blueprint bundle must fail when the bean is constructed. At the same time, a Blueprint bundle must not be required to have any permission needed by the Blueprint Container to perform its tasks.

A Blueprint Container must never use the setAccessibility method on a returned member. Only publicly accessible members must be used. Using a non-publicly accessible member must initiate failure, resulting in the destruction of the container.

121.16.2 Blueprint Bundle

A Blueprint Bundle must have all the permissions required by its code. There is one additional permission required for the Blueprint Bundle. The Blueprint extender will register a Blueprint Container service on behalf of the Blueprint bundle, and the Blueprint bundle must therefore have:

```java
ServicePermission(...BlueprintContainer,[REGISTER])
```

121.17 org.osgi.service.blueprint.container

Blueprint Container Package Version 1.0.

This package defines the primary interface to a Blueprint Container, BlueprintContainer. An instance of this type is available inside a Blueprint Container as an implicitly defined component with the name “blueprintContainer”.

This package also declares the supporting exception types, listener, and constants for working with a Blueprint Container.

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:
121.17.1 Summary

- BlueprintContainer - A Blueprint Container represents the managed state of a Blueprint bundle.
- BlueprintEvent - A Blueprint Event.
- BlueprintListener - A BlueprintEvent Listener.
- ComponentDefinitionException - A Blueprint exception indicating that a component definition is in error.
- Converter - Type converter to convert an object to a target type.
- EventConstants - Event property names used in Event Admin events published by a Blueprint Container.
- NoSuchComponentException - A Blueprint exception indicating that a component does not exist in a Blueprint Container.
- ReifiedType - Provides access to a concrete type and its optional generic type parameters.
- ServiceUnavailableException - A Blueprint exception indicating that a service is unavailable.

121.17.2 public interface BlueprintContainer

A Blueprint Container represents the managed state of a Blueprint bundle. A Blueprint Container provides access to all managed components. These are the beans, services, and service references. Only bundles in the ACTIVE state (and also the STARTING state for bundles awaiting lazy activation) can have an associated Blueprint Container. A given Bundle Context has at most one associated Blueprint Container. A Blueprint Container can be obtained by injecting the predefined "blueprint-container" component id. The Blueprint Container is also registered as a service and its managed components can be queried.

Concurrency: Thread-safe
Provider Type: Consumers of this API must not implement this type

121.17.2.1 public Set<String> getComponentIds()

- Returns the set of component ids managed by this Blueprint Container.

Returns: An immutable Set of Strings, containing the ids of all of the components managed within this Blueprint Container.

121.17.2.2 public Object getComponentInstance(String id)

- id The component id for the requested component instance.

- Returns: A component instance for the component with the specified component id.

Throws: NoSuchComponentException – If no component with the specified component id is managed by this Blueprint Container.

121.17.2.3 public ComponentMetadata getComponentMetadata(String id)

- id The component id for the requested Component Metadata.

- Returns: The Component Metadata object for the component with the specified component id.
Returns The Component Metadata object for the component with the specified component id.

Throws NoSuchComponentException – If no component with the specified component id is managed by this Blueprint Container.

121.17.2.4

```java
public Collection<T> getMetadata(Class<T> type)
```

Type Arguments `<T extends ComponentMetadata`

`<T>` Type of Component Metadata.

type Return all ComponentMetadata objects of the specified Component Metadata type. The supported Component Metadata types are ComponentMetadata (which returns the Component Metadata for all defined manager types), BeanMetadata, ServiceReferenceMetadata (which returns both ReferenceMetadata and ReferenceListMetadata objects), and ServiceMetadata. The collection will include all Component Metadata objects of the requested type, including components that are declared inline.

Returns An immutable collection of Component Metadata objects of the specified type.

121.17.3

```java
public class BlueprintEvent
```

A Blueprint Event.

BlueprintEvent objects are delivered to all registered BlueprintListener services. Blueprint Events must be asynchronously delivered in chronological order with respect to each listener.

In addition, after a Blueprint Listener is registered, the Blueprint extender will synchronously send to this Blueprint Listener the last Blueprint Event for each ready Blueprint bundle managed by this extender. This replay of Blueprint Events is designed so that the new Blueprint Listener can be informed of the state of each Blueprint bundle. Blueprint Events sent during this replay will have the isReplay() flag set. The Blueprint extender must ensure that this replay phase does not interfere with new Blueprint Events so that the chronological order of all Blueprint Events received by the Blueprint Listener is preserved. If the last Blueprint Event for a given Blueprint bundle is DESTROYED, the extender must not send it during this replay phase.

A type code is used to identify the type of event. The following event types are defined:

- CREATING
- CREATED
- DESTROYING
- DESTROYED
- FAILURE
- GRACE_PERIOD
- WAITING

In addition to calling the registered BlueprintListener services, the Blueprint extender must also send those events to the Event Admin service, if it is available.

See Also BlueprintListener, EventConstants

Concurrency Immutable

121.17.3.1 public static final int CREATED = 2

The Blueprint extender has created a Blueprint Container for the bundle. This event is sent after the Blueprint Container has been registered as a service.

121.17.3.2 public static final int CREATING = 1

The Blueprint extender has started creating a Blueprint Container for the bundle.
public static final int DESTROYED = 4
The Blueprint Container for the bundle has been completely destroyed. This event is sent after the Blueprint Container has been unregistered as a service.

public static final int DESTROYING = 3
The Blueprint extender has started destroying the Blueprint Container for the bundle.

public static final int FAILURE = 5
The Blueprint Container creation for the bundle has failed. If this event is sent after a timeout in the Grace Period, the getDependencies() method must return an array of missing mandatory dependencies. The event must also contain the cause of the failure as a Throwable through the getCause() method.

public static final int GRACE_PERIOD = 6
The Blueprint Container has entered the grace period. The list of missing dependencies must be made available through the getDependencies() method. During the grace period, a GRACE_PERIOD event is sent each time the set of unsatisfied dependencies changes.

public static final int WAITING = 7
The Blueprint Container is waiting on the availability of a service to satisfy an invocation on a referenced service. The missing dependency must be made available through the getDependencies() method which will return an array containing one filter object as a String.

public BlueprintEvent(int type, Bundle bundle, Bundle extenderBundle)
type The type of this event.
bundle The Blueprint bundle associated with this event. This parameter must not be null.
extenderBundle The Blueprint extender bundle that is generating this event. This parameter must not be null.
□ Create a simple BlueprintEvent object.

public BlueprintEvent(int type, Bundle bundle, Bundle extenderBundle, String[] dependencies)
type The type of this event.
bundle The Blueprint bundle associated with this event. This parameter must not be null.
extenderBundle The Blueprint extender bundle that is generating this event. This parameter must not be null.
dependencies An array of String filters for each dependency associated with this event. Must be a non-empty array for event types GRACE_PERIOD and WAITING. It is optional for event type FAILURE. Must be null for other event types.
□ Create a BlueprintEvent object associated with a set of dependencies.

public BlueprintEvent(int type, Bundle bundle, Bundle extenderBundle, Throwable cause)
type The type of this event.
bundle The Blueprint bundle associated with this event. This parameter must not be null.
extenderBundle The Blueprint extender bundle that is generating this event. This parameter must not be null.
cause A Throwable object describing the root cause of the event. May be null.
□ Create a BlueprintEvent object associated with a failure cause.

public BlueprintEvent(int type, Bundle bundle, Bundle extenderBundle, String[] dependencies, Throwable cause)
type The type of this event.
bundle The Blueprint bundle associated with this event. This parameter must not be null.

type The type of this event.

dependencies An array of String filters for each dependency associated with this event. Must be a non-empty array for event types GRACE_PERIOD and WAITING. It is optional for event type FAILURE. Must be null for other event types.

cause A Throwable object describing the root cause of this event. May be null.

- Create a BlueprintEvent object associated with a failure cause and a set of dependencies.

121.17.3.12 public BlueprintEvent(BlueprintEvent event, boolean replay)

event The original BlueprintEvent to copy. Must not be null.

replay true if this event should be used as a replay event.

- Create a new BlueprintEvent from the specified BlueprintEvent. The timestamp property will be copied from the original event and only the replay property will be overridden with the given value.

121.17.3.13 public Bundle getBundle()

- Return the Blueprint bundle associated with this event.

Returns The Blueprint bundle associated with this event.

121.17.3.14 public Throwable getCause()

- Return the cause for this FAILURE event.

Returns The cause of the failure for this event. May be null.

121.17.3.15 public String[] getDependencies()

- Return the filters identifying the missing dependencies that caused this event.

Returns The filters identifying the missing dependencies that caused this event if the event type is one of WAITING, GRACE_PERIOD or FAILURE or null for the other event types.

121.17.3.16 public Bundle getExtenderBundle()

- Return the Blueprint extender bundle that is generating this event.

Returns The Blueprint extender bundle that is generating this event.

121.17.3.17 public long getTimestamp()

- Return the time at which this event was created.

Returns The time at which this event was created.

121.17.3.18 public int getType()

- Return the type of this event.

The type values are:

- CREATING
- CREATED
- DESTROYING
- DESTROYED
- FAILURE
- GRACE_PERIOD
- WAITING

Returns The type of this event.
121.17.3.19  public boolean isReplay()

□ Return whether this event is a replay event.

Returns  true if this event is a replay event and false otherwise.

121.17.4  public interface BlueprintListener

A BlueprintEvent Listener.

To receive Blueprint Events, a bundle must register a Blueprint Listener service. After a Blueprint Listener is registered, the Blueprint extender must synchronously send to this Blueprint Listener the last Blueprint Event for each ready Blueprint bundle managed by this extender. This replay of Blueprint Events is designed so that the new Blueprint Listener can be informed of the state of each Blueprint bundle. Blueprint Events sent during this replay will have the isReplay() flag set. The Blueprint extender must ensure that this replay phase does not interfere with new Blueprint Events so that the chronological order of all Blueprint Events received by the Blueprint Listener is preserved. If the last Blueprint Event for a given Blueprint bundle is DESTROYED, the extender must not send it during this replay phase.

See Also  BlueprintEvent

Concurrency  Thread-safe

121.17.4.1  public void blueprintEvent(BlueprintEvent event)

event  The BlueprintEvent.

□ Receives notifications of a Blueprint Event. Implementers should quickly process the event and return.

121.17.5  public class ComponentDefinitionException

extends RuntimeException

A Blueprint exception indicating that a component definition is in error. This exception is thrown when a configuration-related error occurs during creation of a Blueprint Container.

121.17.5.1  public ComponentDefinitionException()

□ Creates a Component Definition Exception with no message or exception cause.

121.17.5.2  public ComponentDefinitionException(String explanation)

explanation  The associated message.

□ Creates a Component Definition Exception with the specified message

121.17.5.3  public ComponentDefinitionException(String explanation, Throwable cause)

explanation  The associated message.

cause  The cause of this exception.

□ Creates a Component Definition Exception with the specified message and exception cause.

121.17.5.4  public ComponentDefinitionException(Throwable cause)

cause  The cause of this exception.

□ Creates a Component Definition Exception with the exception cause.

121.17.6  public interface Converter

Type converter to convert an object to a target type.

Concurrency  Thread-safe
**121.17.6.1**

```java
public boolean canConvert(Object sourceObject, ReifiedType targetType)
```

**Parameters**

- `sourceObject` The source object to convert.
- `targetType` The target type T.

**Returns**

Return if this converter is able to convert the specified object to the specified type.

**true** if the conversion is possible, **false** otherwise.

**121.17.6.2**

```java
public Object convert(Object sourceObject, ReifiedType targetType) throws Exception
```

**Parameters**

- `sourceObject` The source object to convert.
- `targetType` The target type T.

**Returns**

Convert the specified object to an instance of the specified type.

**An instance with a type that is assignable from targetType's raw class**

**Throws**

- `Exception` – If the conversion cannot succeed. This exception should not be thrown when the canConvert method has returned **true**.

**121.17.7**

**public class EventConstants**

Event property names used in Event Admin events published by a Blueprint Container.

Each type of event is sent to a different topic:

```
org/osgi/service/blueprint/container/<event-type>
```

where `<event-type>` can have the values CREATING, CREATED, DESTROYING, DESTROYED, FAILURE, GRACE_PERIOD, or WAITING.

Such events have the following properties:

- `type`
- `event`
- `timestamp`
- `bundle`
  - `bundle.symbolicName`
  - `bundle.id`
  - `bundle.version`
- `extender.bundle.symbolicName`
- `extender.bundle.id`
- `extender.bundle.version`
- `dependencies`
- `cause`

**Concurrency**

Immutable

### 121.17.7.1

**public static final String BUNDLE = "bundle"**

The Blueprint bundle associated with this event. This property is of type Bundle.

### 121.17.7.2

**public static final String BUNDLE_ID = "bundle.id"**

The bundle id of the Blueprint bundle associated with this event. This property is of type Long.

### 121.17.7.3

**public static final String BUNDLE_SYMBOLICNAME = "bundle.symbolicName"**

The bundle symbolic name of the Blueprint bundle associated with this event. This property is of type String.
public static final String BUNDLE_VERSION = "bundle.version"
The bundle version of the Blueprint bundle associated with this event. This property is of type Version.

public static final String CAUSE = "cause"
The cause for a FAILURE event. This property is of type Throwable.

public static final String DEPENDENCIES = "dependencies"
The filters identifying the missing dependencies that caused this event for a FAILURE, GRACE_PERIOD, or WAITING event. This property type is an array of String.

public static final String EVENT = "event"
The BlueprintEvent object that caused this event. This property is of type BlueprintEvent.

public static final String EXTENDER_BUNDLE = "extender.bundle"
The Blueprint extender bundle that is generating this event. This property is of type Bundle.

public static final String EXTENDER_BUNDLE_ID = "extender.bundle.id"
The bundle id of the Blueprint extender bundle that is generating this event. This property is of type Long.

public static final String EXTENDER_BUNDLE_SYMBOLICNAME = "extender.bundle.symbolicName"
The bundle symbolic of the Blueprint extender bundle that is generating this event. This property is of type String.

public static final String EXTENDER_BUNDLE_VERSION = "extender.bundle.version"
The bundle version of the Blueprint extender bundle that is generating this event. This property is of type Version.

public static final String TIMESTAMP = "timestamp"
The time the event was created. This property is of type Long.

public static final String TOPIC_BLUEPRINT_EVENTS = "org/osgi/service/blueprint/container"
Topic prefix for all events issued by the Blueprint Container

public static final String TOPIC_CREATED = "org/osgi/service/blueprint/container/CREATED"
Topic for Blueprint Container CREATED events

public static final String TOPIC_CREATING = "org/osgi/service/blueprint/container/CREATING"
Topic for Blueprint Container CREATING events

public static final String TOPIC_DESTROYED = "org/osgi/service/blueprint/container/DESTROYED"
Topic for Blueprint Container DESTROYED events

public static final String TOPIC_DESTROYING = "org/osgi/service/blueprint/container/DESTROYING"
Topic for Blueprint Container DESTROYING events

public static final String TOPIC_FAILURE = "org/osgi/service/blueprint/container/FAILURE"
Topic for Blueprint Container FAILURE events
121.17.19  public static final String TOPIC_GRACE_PERIOD = "org/osgi/service/blueprint/container/GRACE_PERIOD"
Topic for Blueprint Container GRACE_PERIOD events

121.17.20  public static final String TOPIC_WAITING = "org/osgi/service/blueprint/container/WAITING"
Topic for Blueprint Container WAITING events

121.17.21  public static final String TYPE = "type"
The type of the event that has been issued. This property is of type Integer and can take one of the
values defined in BlueprintEvent.

121.17.8  public class NoSuchComponentException
extends RuntimeException
A Blueprint exception indicating that a component does not exist in a Blueprint Container. This ex-
ception is thrown when an attempt is made to create a component instance or lookup Component
Metadata using a component id that does not exist in the Blueprint Container.

121.17.8.1  public NoSuchComponentException(String msg,String id)
  msg  The associated message.
  id  The id of the non-existent component.
  □  Create a No Such Component Exception for a non-existent component.

121.17.8.2  public NoSuchComponentException(String id)
  id  The id of the non-existent component.
  □  Create a No Such Component Exception for a non-existent component.

121.17.8.3  public String getComponentId()
  □  Returns the id of the non-existent component.
  Returns  The id of the non-existent component.

121.17.9  public class ReifiedType
Provides access to a concrete type and its optional generic type parameters.

Java 5 and later support generic types. These types consist of a raw class with type parameters. This
class models such a Type class but ensures that the type is reified. Reification means that the Type
graph associated with a Java 5 Type instance is traversed until the type becomes a concrete class.
This class is available with the getRawClass() method. The optional type parameters are recursively
represented as Reified Types.

In Java 1.4, a class has by definition no type parameters. This class implementation provides the Rei-
fied Type for Java 1.4 by making the raw class the Java 1.4 class and using a Reified Type based on the
Object class for any requested type parameter.

A Blueprint extender implementations can subclass this class and provide access to the generic type
parameter graph for conversion. Such a subclass must reify the different Java 5 Type instances into
the reified form. That is, a form where the raw Class is available with its optional type parameters as
Reified Types.

Concurrency  Immutable

121.17.9.1  public ReifiedType(Class<? super clazz)
  clazz  The raw class of the Reified Type.
Create a Reified Type for a raw Java class without any generic type parameters. Subclasses can provide the optional generic type parameter information. Without subclassing, this instance has no type parameters.

121.17.9.2 `public ReifiedType getActualTypeArgument(int i)`

- `i` The zero-based index of the requested type parameter.

Return a type parameter for this type. The type parameter refers to a parameter in a generic type declaration given by the zero-based index `i`. For example, in the following example:

```java
Map<String, ? extends Metadata>
```

Type parameter 0 is `String`, and type parameter 1 is `Metadata`.

This implementation returns a Reified Type that has `Object` as class. Any object is assignable to `Object` and therefore no conversion is then necessary. This is compatible with versions of Java language prior to Java 5. This method should be overridden by a subclass that provides access to the generic type parameter information for Java 5 and later.

Returns The ReifiedType for the generic type parameter at the specified index.

121.17.9.3 `public Class<?> getRawClass()`

Return the raw class represented by this type. The raw class represents the concrete class that is associated with a type declaration. This class could have been deduced from the generics type parameter graph of the declaration. For example, in the following example:

```java
Map<String, ? extends Metadata>
```

The raw class is the `Map` class.

Returns The raw class represented by this type.

121.17.9.4 `public int size()`

Return the number of type parameters for this type.

This implementation returns 0. This method should be overridden by a subclass that provides access to the generic type parameter information for Java 5 and later.

Returns The number of type parameters for this type.

121.17.10 `public class ServiceUnavailableException
extends ServiceException`

A Blueprint exception indicating that a service is unavailable. This exception is thrown when an invocation is made on a service reference and a backing service is not available.

121.17.10.1 `public ServiceUnavailableException(String message,String filter)`

- `message` The associated message.
- `filter` The filter used for the service lookup.

Creates a Service Unavailable Exception with the specified message.

121.17.10.2 `public ServiceUnavailableException(String message,String filter,Throwable cause)`

- `message` The associated message.
- `filter` The filter used for the service lookup.
- `cause` The cause of this exception.
Creates a Service Unavailable Exception with the specified message and exception cause.

```
public String getFilter()
```

- Returns the filter expression that a service would have needed to satisfy in order for the invocation to proceed.

  Returns: The failing filter.

### 121.18 org.osgi.service.blueprint.reflect

Blueprint Reflection 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.blueprint.reflect; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.blueprint.reflect; version="[1.0,1.1)"
```

#### 121.18.1 Summary

- **BeanArgument** - Metadata for a factory method or constructor argument of a bean.
- **BeanMetadata** - Metadata for a Bean component.
- **BeanProperty** - Metadata for a property to be injected into a bean.
- **CollectionMetadata** - Metadata for a collection based value.
- **ComponentMetadata** - Metadata for managed components.
- **IdRefMetadata** - Metadata for the verified id of another component managed by the Blueprint Container.
- **MapEntry** - Metadata for a map entry.
- **MapMetadata** - Metadata for a Map based value.
- **Metadata** - Top level Metadata type.
- **NonNullMetadata** - Metadata for a value that cannot null.
- **NullMetadata** - Metadata for a value specified to be null via the `<null>` element.
- **PropsMetadata** - Metadata for a java.util.Properties based value.
- **ReferenceListMetadata** - Metadata for a list of service references.
- **ReferenceMetadata** - Metadata for a reference that will bind to a single matching service in the service registry.
- **RefMetadata** - Metadata for a reference to another component managed by the Blueprint Container.
- **RegistrationListener** - Metadata for a registration listener interested in service registration and unregistration events for a service.
- **ServiceMetadata** - Metadata for a service to be registered by the Blueprint Container when enabled.
- **ServiceReferenceMetadata** - Metadata for a reference to an OSGi service.
- **Target** - A common interface for managed components that can be used as a direct target for method calls.
• ValueMetadata - Metadata for a simple String value that will be type-converted if necessary before injecting.

121.18.2 public interface BeanArgument

Metadata for a factory method or constructor argument of a bean. The arguments of a bean are obtained from BeanMetadata.getArguments(). This is specified by the argument elements of a bean.

Concurrency Thread-safe

121.18.2.1 public int getIndex()

Return the zero-based index into the parameter list of the factory method or constructor to be invoked for this argument. This is determined by specifying the index attribute for the bean. If not explicitly set, this will return -1 and the initial ordering is defined by its position in the BeanMetadata.getArguments() list. This is specified by the index attribute.

Returns The zero-based index of the parameter, or -1 if no index is specified.

121.18.2.2 public Metadata getValue()

Return the Metadata for the argument value. This is specified by the value attribute.

Returns The Metadata for the argument value.

121.18.2.3 public String getValueType()

Return the name of the value type to match the argument and convert the value into when invoking the constructor or factory method. This is specified by the type attribute.

Returns The name of the value type to convert the value into, or null if no type is specified.

121.18.3 public interface BeanMetadata extends Target, ComponentMetadata

Metadata for a Bean component. This is specified by the bean element.

Concurrency Thread-safe

121.18.3.1 public static final String SCOPE_PROTOTYPE = "prototype"

The bean has prototype scope.

See Also getScope()

121.18.3.2 public static final String SCOPE_SINGLETON = "singleton"

The bean has singleton scope.

See Also getScope()

121.18.3.3 public List<BeanArgument> getArguments()

Return the arguments for the factory method or constructor of the bean. This is specified by the child argument elements.

Returns An immutable List of BeanArgument objects for the factory method or constructor of the bean. The List is empty if no arguments are specified for the bean.

121.18.3.4 public String getClassName()

Return the name of the class specified for the bean. This is specified by the class attribute of the bean definition.
Returns The name of the class specified for the bean. If no class is specified in the bean definition, because the a factory component is used instead, then this method will return null.

121.18.3.5 public String getDestroyMethod()
  □ Return the name of the destroy method specified for the bean. This is specified by the destroy-method attribute of the bean definition.

Returns The name of the destroy method specified for the bean, or null if no destroy method is specified.

121.18.3.6 public Target getFactoryComponent()
  □ Return the Metadata for the factory component on which to invoke the factory method for the bean. This is specified by the factory-ref attribute of the bean.

When a factory method and factory component have been specified for the bean, this method returns the factory component on which to invoke the factory method for the bean. When no factory component has been specified this method will return null. When a factory method has been specified for the bean but a factory component has not been specified, the factory method must be invoked as a static method on the bean's class.

Returns The Metadata for the factory component on which to invoke the factory method for the bean or null if no factory component is specified.

121.18.3.7 public String getFactoryMethod()
  □ Return the name of the factory method for the bean. This is specified by the factory-method attribute of the bean.

Returns The name of the factory method of the bean or null if no factory method is specified for the bean.

121.18.3.8 public String getInitMethod()
  □ Return the name of the init method specified for the bean. This is specified by the init-method attribute of the bean definition.

Returns The name of the init method specified for the bean, or null if no init method is specified.

121.18.3.9 public List<BeanProperty> getProperties()
  □ Return the properties for the bean. This is specified by the child property elements.

Returns An immutable List of BeanProperty objects, with one entry for each property to be injected in the bean. The List is empty if no property injection is specified for the bean.

121.18.3.10 public String getScope()
  □ Return the scope for the bean.

Returns The scope for the bean. Returns null if the scope has not been explicitly specified in the bean definition.

See Also SCOPE_SINGLETON, SCOPE_PROTOTYPE

121.18.4 public interface BeanProperty

Metadata for a property to be injected into a bean. The properties of a bean are obtained from BeanMetadata.getProperties(). This is specified by the property elements of a bean. Properties are defined according to the Java Beans conventions.

Concurrency Thread-safe

121.18.4.1 public String getName()
  □ Return the name of the property to be injected. The name follows Java Beans conventions. This is specified by the name attribute.
121.18.4.2 public Metadata getValue()

Return the Metadata for the value to be injected into a bean. This is specified by the value attribute or in inlined text.

Returns The Metadata for the value to be injected into a bean.

121.18.5 public interface CollectionMetadata extends NonNullMetadata

Metadata for a collection based value. Values of the collection are defined by Metadata objects. This Collection Metadata can constrain the values of the collection to a specific type.

Concurrency Thread-safe

121.18.5.1 public Class<?> getCollectionClass()

Return the type of the collection. The possible types are: array (Object[]), Set, and List. This information is specified in the element name.

Returns The type of the collection. Object[] is returned to indicate an array.

121.18.5.2 public List<Metadata> getValues()

Return Metadata for the values of the collection.

Returns A List of Metadata for the values of the collection.

121.18.5.3 public String getValueType()

Return the type specified for the values of the collection. The value-type attribute specified this information.

Returns The type specified for the values of the collection.

121.18.6 public interface ComponentMetadata extends NonNullMetadata

Metadata for managed components. This is the base type for BeanMetadata, ServiceMetadata and ServiceReferenceMetadata.

Concurrency Thread-safe

121.18.6.1 public static final int ACTIVATION_EAGER = 1

The component’s manager must eagerly activate the component.

See Also getActivation()

121.18.6.2 public static final int ACTIVATION_LAZY = 2

The component’s manager must lazily activate the component.

See Also getActivation()

121.18.6.3 public int getActivation()

Return the activation strategy for the component. This is specified by the activation attribute of a component definition. If this is not set, then the default-activation in the blueprint element is used. If that is also not set, then the activation strategy is ACTIVATION_EAGER.

Returns The activation strategy for the component.

See Also ACTIVATION_EAGER, ACTIVATION_LAZY
121.18.6.4  public List<String> getDependsOn()

    □ Return the ids of any components listed in a depends-on attribute for the component.

*Returns*  An immutable List of component ids that are explicitly declared as a dependency, or an empty List if none.

121.18.6.5  public String getId()

    □ Return the id of the component.

*Returns*  The id of the component. The component id can be null if this is an anonymously defined and/or inlined component.

121.18.7  public interface IdRefMetadata extends NonNullMetadata

Metadata for the verified id of another component managed by the Blueprint Container. The id itself will be injected, not the component to which the id refers. No implicit dependency is created.

*Concurrency*  Thread-safe

121.18.7.1  public String GetComponentId()

    □ Return the id of the referenced component. This is specified by the component-id attribute of a component.

*Returns*  The id of the referenced component.

121.18.8  public interface MapEntry

Metadata for a map entry. This type is used by MapMetadata, PropsMetadata and ServiceMetadata.

*Concurrency*  Thread-safe

121.18.8.1  public NonNullMetadata getKey()

    □ Return the Metadata for the key of the map entry. This is specified by the key attribute or element.

*Returns*  The Metadata for the key of the map entry. This must not be null.

121.18.8.2  public Metadata getValue()

    □ Return the Metadata for the value of the map entry. This is specified by the value attribute or element.

*Returns*  The Metadata for the value of the map entry. This must not be null.

121.18.9  public interface MapMetadata extends NonNullMetadata

Metadata for a Map based value.

This is specified by the map element.

*Concurrency*  Thread-safe

121.18.9.1  public List<MapEntry> getEntries()

    □ Return the entries for the map.

*Returns*  An immutable List of MapEntry objects for each entry in the map. The List is empty if no entries are specified for the map.

121.18.9.2  public String getKeyType()

    □ Return the name of the type of the map keys. This is specified by the key-type attribute of the map.
Returns The name of the type of the map keys, or null if none is specified.

121.18.9.3 public String getValueType()

Return the name of the type of the map values. This is specified by the value-type attribute of the map.

Returns The name of the type of the map values, or null if none is specified.

121.18.10 public interface Metadata

Top level Metadata type. All Metadata types extends this base type.

Concurrency Thread-safe

121.18.11 public interfaceNonNullMetadata extends Metadata

Metadata for a value that cannot null. All Metadata subtypes extend this type except for NullMetadata.

This Metadata type is used for keys in Maps because they cannot be null.

Concurrency Thread-safe

121.18.12 public interface NullMetadata extends Metadata

Metadata for a value specified to be null via the <null> element.

Concurrency Thread-safe

121.18.12.1 public static final NullMetadata NULL

Singleton instance of NullMetadata.

121.18.13 public interface PropsMetadata extends NonNullMetadata

Metadata for a java.util.Properties based value.

The MapEntry objects of properties are defined with keys and values of type String.

This is specified by the props element.

Concurrency Thread-safe

121.18.13.1 public List<MapEntry> getEntries()

Return the entries for the properties.

Returns An immutable List of MapEntry objects for each entry in the properties. The List is empty if no entries are specified for the properties.

121.18.14 public interface ReferenceListener

Metadata for a reference listener interested in the reference bind and unbind events for a service reference.

Concurrency Thread-safe

121.18.14.1 public String getBindMethod()

Return the name of the bind method. The bind method will be invoked when a matching service is bound to the reference. This is specified by the bind-method attribute of the reference listener.
Returns The name of the bind method.

121.18.14.2  

```
public Target getListenerComponent()
```

Return the Metadata for the component that will receive bind and unbind events. This is specified by the ref attribute or via an inlined component.

Returns The Metadata for the component that will receive bind and unbind events.

121.18.14.3  

```
public String getUnbindMethod()
```

Return the name of the unbind method. The unbind method will be invoked when a matching service is unbound from the reference. This is specified by the unbind-method attribute of the reference listener.

Returns The name of the unbind method.

121.18.15  

```
public interface ReferenceListMetadata extends ServiceReferenceMetadata
```

Metadata for a list of service references. This is specified by the reference-list element.

Concurrency Thread-safe

121.18.15.1  

```
public static final int USE_SERVICE_OBJECT = 1
```

Reference list values must be proxies to the actual service objects.

See Also getMemberType()

121.18.15.2  

```
public static final int USE_SERVICE_REFERENCE = 2
```

Reference list values must be ServiceReference objects.

See Also getMemberType()

121.18.15.3  

```
public int getMemberType()
```

Return whether the List will contain service object proxies or ServiceReference objects. This is specified by the member-type attribute of the reference list.

Returns Whether the List will contain service object proxies or ServiceReference objects.

See Also USE_SERVICE_OBJECT, USE_SERVICE_REFERENCE

121.18.16  

```
public interface ReferenceMetadata extends Target, ServiceReferenceMetadata
```

Metadata for a reference that will bind to a single matching service in the service registry. This is specified by the reference element.

Concurrency Thread-safe

121.18.16.1  

```
public long getTimeout()
```

Return the timeout for service invocations when a backing service is is unavailable. This is specified by the timeout attribute of the reference.

Returns The timeout, in milliseconds, for service invocations when a backing service is is unavailable.

121.18.17  

```
public interface RefMetadata extends Target, NonNullMetadata
```

Metadata for a reference to another component managed by the Blueprint Container.
**Concurrency**  Thread-safe

121.18.17  public String getComponentId()

- Return the id of the referenced component. This is specified by the component-id attribute of a component.

  *Returns*  The id of the referenced component.

121.18  **public interface RegistrationListener**

Implementation of a context to receive service registration and unregistration events. The registration listener is called with the initial state of the service when the registration listener is actuated.

**Concurrency**  Thread-safe

121.18.18 1

121.18.18.1  public Target getListenerComponent()

- Return the Metadata for the component that will receive registration and unregistration events. This is specified by the ref attribute or via an inlined component.

  *Returns*  The Metadata for the component that will receive registration and unregistration events.

121.18.18.2  public String getRegistrationMethod()

- Return the name of the registration method. The registration method will be invoked when the associated service is registered with the service registry. This is specified by the registration-method attribute of the registration listener.

  *Returns*  The name of the registration method.

121.18.18.3  public String getUnregistrationMethod()

- Return the name of the unregistration method. The unregistration method will be invoked when the associated service is unregistered from the service registry. This is specified by the unregistration-method attribute of the registration listener.

  *Returns*  The name of the unregistration method.

121.18.19  **public interface ServiceMetadata**

Extends ComponentMetadata

Metadata for a service to be registered by the Blueprint Container when enabled. This is specified by the service element.

**Concurrency**  Thread-safe

121.18.19.1  public static final int AUTO_EXPORT_ALL_CLASSES = 4

Advertise all Java classes and interfaces in the component instance type as service interfaces.

  *See Also*  getAutoExport()

121.18.19.2  public static final int AUTO_EXPORT_CLASS_HIERARCHY = 3

Advertise all Java classes in the hierarchy of the component instance type as service interfaces.

  *See Also*  getAutoExport()

121.18.19.3  public static final int AUTO_EXPORT_DISABLED = 1

Do not auto-detect types for advertised service interfaces
public static final int AUTO_EXPORT_INTERFACES = 2

Advertise all Java interfaces implemented by the component instance type as service interfaces.

See Also getAutoExport()

public int getAutoExport()

Return the auto-export mode for the service. This is specified by the auto-export attribute of the service.

Returns The auto-export mode for the service.

See Also AUTO_EXPORT_DISABLED, AUTO_EXPORT_INTERFACES, AUTO_EXPORT_CLASS_HIERARCHY, AUTO_EXPORT_ALL_CLASSES

public List<String> getInterfaces()

Return the type names of the interfaces that the service should be advertised as supporting. This is specified in the interface attribute or child interfaces element of the service.

Returns An immutable List of String for the type names of the interfaces that the service should be advertised as supporting. The List is empty if using auto-export or no interface names are specified for the service.

public int getRanking()

Return the ranking value to use when advertising the service. If the ranking value is zero, the service must be registered without a service-ranking service property. This is specified by the ranking attribute of the service.

Returns The ranking value to use when advertising the service.

public Collection<RegistrationListener> getRegistrationListeners()

Return the registration listeners to be notified when the service is registered and unregistered with the framework. This is specified by the registration-listener elements of the service.

Returns An immutable Collection of RegistrationListener objects to be notified when the service is registered and unregistered with the framework. The Collection is empty if no registration listeners are specified for the service.

public Target getServiceComponent()

Return the Metadata for the component to be exported as a service. This is specified inline or via the ref attribute of the service.

Returns The Metadata for the component to be exported as a service.

public List<MapEntry> getServiceProperties()

Return the user declared properties to be advertised with the service. This is specified by the service-properties element of the service.

Returns An immutable List of MapEntry objects for the user declared properties to be advertised with the service. The List is empty if no service properties are specified for the service.

public interface ServiceReferenceMetadata extends ComponentMetadata

Metadata for a reference to an OSGi service. This is the base type for ReferenceListMetadata and ReferenceMetadata.

Concurrency Thread-safe
121.18.20.1  public static final int AVAILABILITY_MANDATORY = 1
A matching service is required at all times.
See Also  getAvailability()

121.18.20.2  public static final int AVAILABILITY_OPTIONAL = 2
A matching service is not required to be present.
See Also  getAvailability()

121.18.20.3  public int getAvailability()

Get whether or not a matching service is required at all times. This is specified in the availability attribute of the service reference.

Returns  Whether or not a matching service is required at all times.
See Also  AVAILABILITY_MANDATORY, AVAILABILITY_OPTIONAL

121.18.20.4  public String getComponentName()

Return the value of the component-name attribute of the service reference. This specifies the id of a component that is registered in the service registry. This will create an automatic filter, appended with the filter if set, to select this component based on its automatic id attribute.

Returns  The value of the component-name attribute of the service reference or null if the attribute is not specified.

121.18.20.5  public String getFilter()

Return the filter expression that a matching service must match. This is specified by the filter attribute of the service reference.

Returns  The filter expression that a matching service must match or null if a filter is not specified.

121.18.20.6  public String getInterface()

Return the name of the interface type that a matching service must support. This is specified in the interface attribute of the service reference.

Returns  The name of the interface type that a matching service must support or null when no interface name is specified.

121.18.20.7  public Collection<ReferenceListener> getReferenceListeners()

Return the reference listeners to receive bind and unbind events. This is specified by the reference-listener elements of the service reference.

Returns  An immutable Collection of ReferenceListener objects to receive bind and unbind events. The Collection is empty if no reference listeners are specified for the service reference.

121.18.21  public interface Target
        extends NonNullMetadata
        A common interface for managed components that can be used as a direct target for method calls. These are bean, reference, and ref, where the ref must refer to a bean or reference component.
        See Also  BeanMetadata, ReferenceMetadata, RefMetadata
        Concurrency  Thread-safe

121.18.22  public interface ValueMetadata
        extends NonNullMetadata
        Metadata for a simple String value that will be type-converted if necessary before injecting.
Concurrency  Thread-safe

121.18.22.1  public String getStringValue()

Return the unconverted string representation of the value. This is specified by the value attribute or text part of the value element.

Returns  The unconverted string representation of the value.

121.18.22.2  public String getType()

Return the name of the type to which the value should be converted. This is specified by the type attribute.

Returns  The name of the type to which the value should be converted or null if no type is specified.

121.19  References

[1]  Spring Framework
http://www.springsource.org/

[2]  Spring Dynamic Modules
http://www.springsource.org/osgi

http://code.google.com/p/google-guice/

[4]  Picoccontainer
http://www.picoccontainer.org/

http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138795.html

[6]  XML Namespaces
http://www.w3.org/TR/REC-xml-names

[7]  Properties format
http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Properties.html#load%28java.io.InputStream%29

[8]  XML Schema
http://www.w3.org/XML/Schema

[9]  OSGi XML Schemas
http://www.osgi.org/Specifications/HomePage
122 Remote Service Admin Service Specification

Version 1.0

122.1 Introduction

The OSGi Core Release 6 framework specifies a model where bundles can use distributed services. The basic model for OSGi remote services is that a bundle can register services that are exported to a communication Endpoint and use services that are imported from a communication Endpoint. However, the remote services chapter does not explain what services are exported and/or imported; it leaves such decisions to the distribution provider. The distribution provider therefore performs multiple roles and cannot be leveraged by other bundles in scenarios that the distribution provider had not foreseen.

The primary role of the distribution provider is purely mechanical; it creates Endpoints and registers service proxies and enables their communication. The second role is about the policies around the desired topology. The third role is discovery. To establish a specific topology it is necessary to find out about exported services in other frameworks.

This specification therefore defines an API for the distribution provider and discovery of services in a network. A management agent can use this API to provide an actual distribution policy. This management agent, called the Topology Manager, can control the export and import of services delegating the intrinsic knowledge of the low level details of communication protocols, proxying of services, and discovering services in the network to services defined in this specification.

This specification is an extension of the Remote Service chapter, see chapter Remote Services on page 23. Though some aspects are repeated in this specification, a full understanding of the Remote Services chapter is required for full understanding of this document.

122.1.1 Essentials

- **Simple** - Make it as simple as possible for a Topology Manager to implement distribution policies.
- **Inform** - Provide a mechanism to inform other parties about created and removed Endpoints.
- **Configuration** - Allow bundles to describe Endpoints as a bundle resource that are provided to the Distribution Provider.
- **Selective** - Not all parties are interested in all services. Endpoint registries must be able to express the scope of services they are interested in.
- **Multiple** - Allow the collaboration of multiple Topology Managers, Remote Service Admin services, and Discovery Providers.
- **Dynamic** - Allow the dynamic discovery of Endpoints.
- **Federated** - Enable a global view of all available services in a distributed environment.
122.1.2 Entities

- **Remote Service Admin** - An implementation of this specification provides the mechanisms to import and export services through a set of configuration types. The Remote Service Admin service is a passive Distribution Provider, not taking any action to export or import itself.
- **Topology Manager** - The Topology Manager provides the policy for importing and exporting services through the Remote Service Admin service.
- **Endpoint** - An Endpoint is a communications access mechanism to a service in another framework, a (web) service, another process, or a queue or topic destination, etc., requiring some protocol for communications.
- **Endpoint Description** - A properties based description of an Endpoint. Endpoint Descriptions can be exchanged between different frameworks to create connections to each other’s services. Endpoint Descriptions can also be created to Endpoints not originating in an OSGi Framework.
- **Endpoint Description Provider** - A party that can inform others about the existence of Endpoints.
- **Endpoint Listener** - A listener service that receives updates of Endpoints that match its scope. This Endpoint Listener is used symmetrically to implement a federated registry. The Topology Manager can use it to notify interested parties about created and removed Endpoints, as well as to receive notifications from other parties, potentially remote, about their available Endpoints.
- **Remote Service Admin Listener** - A listener service that is informed of all the primitive actions that the Remote Service Admin performs like importing and exporting as well as errors.
- **Endpoint Configuration Extender** - A bundle that can detect configuration data describing an Endpoint Description in a bundle resource, using the extender pattern.
- **Discovery** - An Endpoint Listener that detects the Endpoint Descriptions through some discovery protocol.
- **Cluster** - A group of computing systems that closely work together, usually in a fast network.

Figure 122.1 Remote Service Admin Entities

122.1.3 Synopsis

Topology Managers are responsible for the distribution policies of an OSGi framework. To implement a policy, a Topology Manager must be aware of the environment, for this reason, it can register:

- Service listeners to detect services that can be exported according to the Remote Services chapter.
- Listener and Find Hook services to detect bundles that have an interest in specific services that potentially could be imported.
Remote Service Admin Service Specification Version 1.0

- A Remote Service Admin Listener service to detect the activity of other Topology Managers.
- An Endpoint Listener service to detect Endpoints that are made available through discovery protocols, configuration data, or other means.

Using this information, the manager implements a topology using the Remote Service Admin service. A Topology Manager that wants to export a service can create an Export Registration by providing one or more Remote Service Admin services a Service Reference plus a Map with the required properties. A Remote Service Admin service then creates a number of Endpoints based on the available configuration types and returns a collection of ExportRegistration objects. A collection is returned because a single service can be exported to multiple Endpoints depending on the available configuration type properties.

Each Export Registration is specific for the caller and represents an existing or newly created Endpoint. The Export Registration associates the exported Service Reference with an Endpoint Description. If there are problems with the export operation, the Remote Service Admin service reports these on the Export Registration objects. That is, not all the returned Export Registrations have to be valid.

An Endpoint Description is a property based description of an Endpoint. Some of these properties are defined in this specification, other properties are defined by configuration types. These configuration types must follow the same rules as the configuration types defined in the Remote Services chapter. Remote Service Admin services that support the configuration types in the Endpoint Description can import a service from that Endpoint solely based on that Endpoint Description.

In similar vein, the Topology Manager can import a service from a remote system by creating an Import Registration out of an Endpoint Description. The Remote Service Admin service then registers a service that is a proxy for the remote Endpoint and returns an ImportRegistration object. If there are problems with the import, the Remote Service Admin service that cannot be detected early, then the Remote Service Admin service reports these on the returned ImportRegistration object.

For introspection, the Remote Service Admin can list its current set of Import and Export References so that a Topology Manager can get the current state. The Remote Service Admin service also informs all Topology Managers and observers of the creation, deletion, and errors of Import and Export Registrations through the Remote Service Admin Listener service. Interested parties like the Topology Manager can register such a service and will be called back with the initial state as well as any subsequent changes.

An important aspect of the Topology Manager is the distributed nature of the scenarios it plays an orchestrating role in. A Topology Manager needs to be aware of Endpoints in the network, not just the ones provided by Remote Service Admin services in its local framework. The Endpoint Listener service is specified for this purpose. This service is provided for both directions, symmetrically. That is, it is used by the Topology Manager to inform any observers about the existence of Endpoints that are locally available, as well as for parties that represent a discovery mechanism. For example Endpoints available on other systems, Endpoint Descriptions embedded in resources in bundles, or Endpoint Descriptions that are available in some other form.

Endpoint Listener services are not always interested in the complete set of available Endpoints because this set can potentially be very large. For example, if a remote registry like [5] UDDI is used then the number of Endpoints can run into the thousands or more. An Endpoint Listener service can therefore scope the set of Endpoints with an OSGi LDAP style filter. Parties that can provide information about Endpoints must only notify Endpoint Listener services when the Endpoint Description falls within the scope of the Endpoint Listener service. Parties that use some discovery mechanism can use the scope to trigger directed searches across the network.
122.2 Actors

The OSGi Remote Services specification is about the distribution of services. OSGi Core Release 6 does not outline the details of how the distribution provider knows the desired topology; this policy aspect is left up to implementations. In many situations, this is a desirable architecture because it provides freedom of implementation to the distribution provider. However, such an architecture does not enable a separation of the mechanisms and policy. Therefore, this Remote Service Admin specification provides an architecture that enables a separate bundle from the distribution provider to define the topology. It splits the responsibility of the Remote Service specification in a number of roles. These roles can all have different implementations but they can collaborate through the services defined in this specification. These roles are:

- **Topology Managers** - Topology Managers are the (anonymous) players that implement the policies for distributing services; they are closely aligned with the concept of an OSGi management agent. It is expected that Topology Managers will be developed for scenarios like import/export all applicable services, configuration based imports- and exports, and scenarios like fail-over, load-balancing, as well as standards like domain managers for the Service Component Architecture (SCA).
- **Remote Service Admin** - The Remote Service Admin service provides the basic mechanism to import and export services. This service is policy free; it will not distribute services without explicitly being told so. A OSGi framework can host multiple Remote Service Admin services that, for example, support different configuration types.
- **Discovery** - To implement a distribution policy, a Topology Manager must be aware of what Endpoints are available. This specification provides an abstraction of a federated Endpoint registry. This registry can be used to both publish as well as consume Endpoints from many different sources. The federated registry is defined for local services but is intended to be used with standard and proprietary service discovery protocols. The federated registry is implemented with the Endpoint Listener service.

These roles are depicted in Figure 122.2 on page 342.

Figure 122.2 Roles

122.3 Topology Managers

Distributed processing has become mainstream because of the massive scale required for Internet applications. Only with distributed architectures is it possible to scale systems to Internet size with hundreds of millions of users. To allow a system to scale, servers are grouped in clusters where they can work in unison or geographically dispersed in even larger configurations. The distribution of the work-load is crucial for the amount of scalability provided by an architecture and often has domain specific dispatching techniques. For example, the hash of a user id can be used to select the correct profile database server. In this fast moving world it is very unlikely that a single architecture or distribution policy would be sufficient to satisfy many users. It is therefore that this specification
separates the how from the what. The complex mechanics of importing and exporting services are managed by a Remote Service Admin service (the how) while the different policies are implemented by Topology Managers (the what). This separation of concerns enables the development of Topology Managers that can run on many different systems, providing high user functionality. For example, a Topology Manager could implement a fail-over policy where some strategic services are redirected when their connections fail. Other Topology Managers could use a discovery protocol like SLP to find out about other systems in a cluster and automatically configure the cluster.

The key value of this architecture is demonstrated by the example of an SCA domain controller. An SCA domain controller receives a description of a domain (a set of systems and modules) and must ensure that the proper connections are made between the participating SCA modules. By splitting the roles, an SCA domain manager can be developed that can run on any compatible Remote Service Admin service implementation.

### 122.3.1 Multiple Topology Managers

There is no restriction on the number of Topology Managers, nor is there a restriction on the number of Remote Service Admin service implementations. It is up to the deployer of the OSGi framework to select the appropriate set of these service implementations. It is the responsibility of the Topology Managers to listen to the Remote Service Admin Listener and track Endpoints created and deleted by other Topology Managers and act appropriately.

### 122.3.2 Example Use Cases

#### 122.3.2.1 Promiscuous Policy

A cluster is a set of machines that are connected in a network. The simplest policy for a Topology Manager is to share exported services in such a cluster. Such a policy is very easy to implement with the Remote Services Admin service. In the most basic form, this Topology Manager would use some multicast protocol to communicate with its peers. These peers would exchange EndpointDescription objects of exported services. Each Topology Manager would then import any exported service.

This scenario can be improved by separating the promiscuous policy from the discovery. Instead of embedding the multicast protocol, a Topology manager could use the Endpoint Listener service. This service allows the discovery of remote services. At the same time, the Topology Manager could tell all other Endpoint Listener services about the services it has created, allowing them to be used by others in the network.

Splitting the Topology Manager and discovery in two bundles allows different implementations of the discovery bundle, for example, to use different protocols.

#### 122.3.2.2 Fail Over

A more elaborate scheme is a fail-over policy. In such a policy a service can be replaced by a service from another machine. There are many ways to implement such a policy, an simple example strategy is provided here for illustration.

A Fail-Over Topology Manager is given a list of stateless services that require fail-over, for example through the Configuration Admin Service Specification on page 63. The Fail-Over Manager tracks the systems in the its cluster that provide such services. This tracking can use an embedded protocol or it can be based on the Endpoint Listener service model.

In the Fail-Over policy, the fail-over manager only imports a single service and then tracks the error status of the imported service through the Remote Service Admin Listener service. If it detects the service is becoming unavailable, it closes the corresponding Import Registration and imports a service from an alternative system instead. In Figure 122.3, there are 4 systems in a cluster. The topology/fail-over manager ensures that there is always one of the services in system A, B, or C available in D.
There are many possible variations on this scenario. The managers could exchange load information, allowing the service switch to be influenced by the load of the target systems. The important aspect is that the Topology Manager can ignore the complex details of discovery protocols, communication protocols, and service proxying and instead focus on the topology.

### 122.4 Endpoint Description

An *Endpoint* is a point of rendezvous of distribution providers. It is created by an exporting distribution provider or some other party, and is used by importing distribution providers to create a connection. An *Endpoint Description* describes an Endpoint in such a way that an importing Remote Service Admin service can create this connection if it recognizes the *configuration type* that is used for that Endpoint. The configuration type consists of a name and a set of properties associated with that name.

The core concept of the Endpoint Description is a Map of properties. The structure of this map is the same as service properties, and the defined properties are closely aligned with the properties of an imported service. An EndpointDescription object must only consist of the data types that are supported for service properties. This makes the property map serializable with many different mechanisms. The EndpointDescription class provides a convenient way to access the properties in a type safe way.

An Endpoint Description has case insensitive keys, just like the Service Reference's properties.

The properties map must contain all the prescribed service properties of the exported service after intents have been processed, as if the service was registered as an imported service. That is, the map must not contain any properties that start with `service.exported.*` but it must contain the `service.imported.*` variation of these properties. The Endpoint Description must reflect the imported service properties because this simplifies the use of filters from the service hooks. Filters applied to the Endpoint Description can then be the same filters as applied by a bundle to select an imported service from the service registry.

The properties that can be used in an Endpoint Description are listed in Table 122.1. The RemoteConstants class contains the constants for all of these property names.

<table>
<thead>
<tr>
<th>Endpoint Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>service.exported.*</code></td>
<td></td>
<td>Must not be set</td>
</tr>
<tr>
<td><code>service.imported</code></td>
<td>*</td>
<td>Must always be set to some value. See <code>SERVICE_IMPORTED</code>.</td>
</tr>
<tr>
<td>Endpoint Property Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>objectClass</td>
<td>String[]</td>
<td>Must be set to the value of service.exported.interfaces, of the exported service after expanding any wildcards. Though this property will be overridden by the framework for the corresponding service registration, it must be set in the Endpoint Description to simplify the filter matching. These interface names are available with the getInterfaces() method.</td>
</tr>
<tr>
<td>service.intents</td>
<td>String+</td>
<td>Intents implemented by the exporting distribution provider and, if applicable, the exported service itself. Any qualified intents must have their expanded form present. These expanded intents are available with the getIntents() method. See SERVICE_INTENTS.</td>
</tr>
<tr>
<td>endpoint.service.id</td>
<td>Long</td>
<td>The service id of the exported service. Can be absent or 0 if the corresponding Endpoint is not for an OSGi service. The remote service id is available as getServiceId(). See also ENDPOINT_SERVICE_ID.</td>
</tr>
<tr>
<td>endpoint.framework.uuid</td>
<td>String</td>
<td>A universally unique id identifying the instance of the exporting framework. Can be absent if the corresponding Endpoint is not for an OSGi service. See Framework UUID on page 347. The remote framework UUID is available with the getFrameworkUUID() method. See also ENDPOINT_FRAMEWORK_UUID.</td>
</tr>
<tr>
<td>endpoint.id</td>
<td>String</td>
<td>The Id for this Endpoint, can never be null. This information is available with the getId(). See Endpoint Id on page 347 and also ENDPOINT_ID.</td>
</tr>
<tr>
<td>endpoint.package version.&lt;package-name&gt;</td>
<td>String</td>
<td>The Java package version for the embedded &lt;package&gt;. For example, the property endpoint.package.version.com.acme=1.3 describes the version for the com.acme package. The version for a package can be obtained with the getPackageVersion(String). The version does not have to be set, if not set, the value must be assumed to be 0.</td>
</tr>
<tr>
<td>service.imported.configs</td>
<td>String+</td>
<td>The configuration types that can be used to implement the corresponding Endpoint. This property maps to the corresponding property in the Remote Services chapter. This property can be obtained with the getConfigurationTypes() method. The Export Registration has all the possible configuration types, where the Import Registration reports the configuration type actually used. SERVICE_IMPORTED_CONFIGS.</td>
</tr>
<tr>
<td>&lt;config&gt;*</td>
<td>*</td>
<td>Where &lt;config&gt; is one of the configuration type names listed in service.imported.configs. The content of these properties must be valid for creating a connection to the Endpoint in another framework. That is, any locally readable URLs from bundles must be converted in such a form that they can be read by the importing framework. How this is done is configuration type specific.</td>
</tr>
</tbody>
</table>
The EndpointDescription class has a number of constructors that make it convenient to instantiate it for different purposes:

- `EndpointDescription(Map)` - Instantiate the Endpoint Description from a Map object.
- `EndpointDescription(ServiceReference,Map)` - Instantiate an Endpoint Description based on a Service Reference and a Map. The base properties of this Endpoint Description are the Service Reference properties but the properties in the given Map must override any of their case variants in the Service Reference. This allows the construction of an Endpoint Description from an exportable service while still allowing overrides of specific properties by the Topology Manager.

The Endpoint Description must use the allowed properties as given in Table 122.1 on page 344. The Endpoint Description must automatically skip any service.exported.* properties.

The Endpoint Description provides the following methods to access the properties in a more convenient way:

- `getInterfaces()` - Answers a list of Java interface names. These are the interfaces under which the services must be registered. These interface names can also be found at the objectClass property. A service can only be imported when there is at least one Java interface name available.
- `getConfigurationTypes()` - Answer the configuration types that are used for exporting this Endpoint. The configuration types are associated with a number of properties.
- `getId()` - Returns an Id uniquely identifying an Endpoint. The syntax of this Id should be defined in the specification for the associated configuration type. Two Endpoint Descriptions with the same Id describe the same Endpoint.
- `getFrameworkUUID()` - Get a Universally Unique Identifier (UUID) for the framework instance that has created the Endpoint, Framework UUID on page 347.
- `getServiceId()` - Get the service id for the framework instance that has created the Endpoint. If there is no service on the remote side the value must be 0.
- `getPackageVersion(String)` - Get the version for the given package.
- `getIntents()` - Get the list of specified intents.
- `getProperties()` - Get all the properties.

Two Endpoint Descriptions are deemed equal when their Endpoint Id is equal. The Endpoint Id is a mandatory property of an Endpoint Description, it is further described at Endpoint Id on page 347. The hash code is therefore also based on the Endpoint Id.

122.4.1 Validity

A valid Endpoint Description must at least satisfy the following assertions:

- It must have a non-null Id that uniquely identifies the Endpoint
- It must at least have one Java interface name
- It must at least have one configuration type set
- Any version for the packages must have a valid version syntax.

122.4.2 Mutability

An EndpointDescription object is immutable and with all final fields. It can be freely used between different threads.
122.4.3 Endpoint Id

An Endpoint Id is an opaque unique identifier for an Endpoint. There is no syntax defined for this string except that white space at the beginning and ending must be ignored. The actual syntax for this Endpoint Id must be defined by the actual configuration type.

Two Endpoint Descriptions are deemed identical when their Endpoint Id is equal. The Endpoint Ids must be compared as string compares with leading and trailing spaces removed. The Endpoint Description class must use the String class’ hashCode from the Endpoint Id as its own hashCode.

122.4.4 Framework UUID

Each framework registers its services with a service id that is only unique for that specific framework. The OSGi framework is not a singleton, making it possible that a single VM process holds multiple OSGi frameworks. Therefore, to identify an OSGi service uniquely it is necessary to identify the framework that has registered it. This identifier is a Universally Unique IDentifier (UUID) that is set for each framework. This UUID is contained in the following framework property:

```
org.osgi.framework.uuid
```

If an Endpoint Description has no associated OSGi service then the UUID of that Endpoint Description must not be set and its service id must be 0.

The deployer should ensure that this property is set properly before the framework is started as either a framework property in the launch, a system property, or the use of a framework implementation that automatically sets this property. If no such property is set, a Remote Service Admin must:

1. Start a synchronized block on the class literal String object `org.osgi.framework.uuid`. This can be done like:

   ```
synchronized("org.osgi.framework.uuid") { ... }
```

2. Check if the framework UUID is set in the Framework properties, if so, use this one and exit the synchronized block.
3. Create a new UUID.
4. Set the UUID property in the System properties.
5. Leave the synchronized block.

These steps ensure that the same UUID is consistently used for all exported services. The Remote Service Admin implementation must have the proper permissions to read and write the System properties if the UUID is not set.

The framework UUID must be constructed according to the `java.util.UUID` class in the form returned from its `toString` method.

A local Endpoint Description will have its framework UUID set to the local framework. This makes it straightforward to filter for Endpoint Descriptions that are describing local Endpoints or that describe remote Endpoints. For example, a manager can take the filter from a listener and ensure that it is only getting remote Endpoint Descriptions:

```
(&
 (!
   {service.remote.framework.uuid
    =72dc5fd9-5f8f-4f8f-9821-9ebb433a5b72}
  )
  {objectClass=org.osgi.service.log.LogService}
)
```
Where 72dc5fd9-5f8f-4f8f-9821-9eb21433a5b72 is the UUID of the local framework. A discovery bundle can register the following filter in its scope to receive all locally generated Endpoints:

```
(service.remote.framework.uuid
 =72dc5fd9-5f8f-4f8f-9821-9eb21433a5b72)
```

### 122.4.5 Resource Containment

Configuration types can, and usually do, use URLs to point to local resources describing in detail the Endpoint parameters for specific protocols. However, the purpose of an Endpoint Description is to describe an Endpoint to a remote system. This implies that there is some marshalling process that will transfer the Endpoint Description to another process. This other process is unlikely to be able to access resource URLs. Local bundle resource URLs are only usable in the framework that originates them but even HTTP based URLs can easily run into problems due to firewalls or lack of routing.

Therefore, the properties for a configuration type should be stored in such a way that the receiving process can access them. One way to achieve this is to contain the configuration properties completely in the Endpoint Description and ensure they only use the basic data types that the remote services chapter in the core requires every Distribution Provider to support.

The Endpoint Description XML format provides an `xml` element that is specifically added to make it easy to embed XML based configuration documents. The XML Schema is defined in `Endpoint Description Extender Format` on page 358.

### 122.5 Remote Service Admin

The Remote Service Admin service abstracts the core functionality of a distribution provider: exporting a service to an Endpoint and importing services from an Endpoint. However, in contrast with the distribution provider of the Remote Services specification, the Remote Service Admin service must be told explicitly what services to import and export.

#### 122.5.1 Exporting

An exportable service can be exported with the `exportService(ServiceReference,Map)` method. This method creates a number of Endpoints by inspecting the merged properties from the Service Reference and the given Map. Any property in the Map overrides the Service Reference properties, regardless of case. That is, if the map contains a key then it will override any case variant of this key in the Service Reference. However, if the Map contains the `objectClass` or `service.id` property key in any case variant, then these properties must not override the Service Reference's value.

The Remote Service Admin service must interpret the merged properties according to the Remote Services chapter. This means that it must look at the following properties (as defined in chapter Remote Services on page 23):

- `service.exported.configs` - (String+) A list of configuration types that should be used to export this service. Each configuration type represents the configuration parameters for an Endpoint. A Remote Service Admin service should create an Endpoint for each configuration type that it supports and ignore the types it does not recognize. If this property is not set, then the Remote Service Admin implementation must choose a convenient configuration type that then must be reported on the Endpoint Description with the `service.imported.configs` associated with the returned Export Registration.
- `service.exported.intents` - (String+) A list of intents that the Remote Service Admin service must implement to distribute the given service.
- `service.exported.intents.extra` - (String+) This property is merged with the `service.exported.intents` property.
- service.exported.interfaces - (String+) This property must be set; it marks this service for export and defines the interfaces. The list members must all be contained in the types listed in the objectClass service property from the Service Reference. The single value of an asterisk ('*') indicates all interfaces in the registration's objectClass property and ignore the classes. Being able to set this property outside the Service Reference implies that the Topology Manager can export any registered service, also services not specifically marked to be exported.

- service.intents - (String+) A list of intents that this service has implemented.

A Topology Manager cannot remove properties, null is invalid as a property value.

The Remote Service Admin returns a collection of ExportRegistration objects. This collection must contain an entry for each configuration type the Remote Service Admin has recognized. Unrecognized configuration types must be ignored. However, it is possible that this list contains invalid registrations, see Invalid Registrations on page 352.

If a Service was already exported then the Remote Service Admin must still return a new ExportRegistration object that is linked with the earlier registrations. That is, an Endpoint can be shared between multiple Export Registrations. The Remote Service Admin service must ensure that the corresponding Endpoint remains available as long as there is at least one open Export Registration for that Endpoint.

For each successful creation of an export registration, the Remote Service Admin service must publish an EXPORT_REGISTRATION event, see Events on page 357. This event must be emitted, even if the Endpoint already existed and is thus shared with another Export Registration. If the creation of an Endpoint runs into an error, an EXPORT_ERROR event must be emitted.

Each valid Export Registration corresponds to an Endpoint for the given service. This Endpoint must remain active until all of the Export Registrations are closed that share this Endpoint.

The Endpoint can now be published so that other processes or systems can import this Endpoint. To aid with this import, the Export Registration has a getExportReference() method that returns an ExportReference object. This reference provides the following information:

- getExportedEndpoint() - This is the associated Endpoint Description. This Endpoint Description is a properties based description of an Endpoint. The property keys and their semantics are outlined in Endpoint Description on page 344. It can be used to inform other systems of the availability of an Endpoint.

- getExportedService() - The Service Reference to the exported service.

Both methods must return null when the associated Export Registration is closed.

A Distribution Provider that recognizes the configuration type in an Endpoint can create a connection to an Endpoint on other systems as long as firewalls and networks permit. The Endpoint Description can therefore be communicated to other systems to announce the availability of an Endpoint. The Topology Manager can optionally announce the availability of an Endpoint to the Endpoint Listener services, see Discovery on page 353. The decision to announce the availability of an Endpoint is one of the policies that is provided by a specific Topology Manager.

The Export Registrations remain open until:

- Explicitly closed by the Topology Manager, or
- The Remote Service Admin service is no longer used by the Topology Manager that created the Export Registration.

If the Remote Service Admin service can no longer maintain the corresponding Endpoint due to failures than these should be reported through the events. However, the registrations should remain open until explicitly closed by the Topology Manager.

See Registration Life Cycle on page 351 for more information.

The Export Registrations are not permanent; persistence is in the realm of the Topology Manager.
122.5.2 Importing

To import a service, a Topology Manager must have an Endpoint Description that describes the Endpoint the imported service should connect to. With this Endpoint Description, a Remote Service Admin service can then import the corresponding Endpoint. A Topology Manager can obtain these Endpoint Descriptions through internal configuration; it can use the discovery model enabled by the Endpoint Listener service, see Discovery on page 353, or some alternate means.

A service can be imported with the Remote Service Admin importService(EndpointDescription) method. This method takes an Endpoint Description and picks one of the embedded configuration types to establish a connection with the corresponding Endpoint to create a local service proxy. This proxy can then be mapped to either a remote OSGi service or an alternative, for example a web service. In certain cases the service proxy can be lazy, only verifying the reachability of the Endpoint when it is actually invoked for the first time. This implies that a service proxy can block when invoked until the proper communication setup has taken place.

If the Remote Service Admin service does not recognize any of the configuration types then it must return null. If there are multiple configuration types recognized then the Remote Service Admin is free to select any one of the recognized types.

If an Endpoint was already imported as a service proxy, then the Remote Service Admin service must return a new Import Registration that is associated with the existing service proxy/Endpoint combination. The Remote Service Admin service must ensure that the imported service proxy remains available as long as there is at least one open Import Registration that refers to it and the corresponding remote Endpoint is still valid.

The Remote Service Admin service must ensure that service properties are according to the Remote Services chapter for an imported service. This means that it must register the following properties:

- service.imported - (*) Must be set to any value.
- service.imported.configs - (String+) The configuration information used to import this service. Any associated properties for this configuration types must be properly mapped to the importing system. For example, a URL in these properties must point to a valid resource when used in the importing framework, see Resource Containment on page 348. Multiple configuration types can be listed if they are synonyms for exactly the same Endpoint that is used to export this service.
- service.intents - (String+) The Remote Service Admin must set this property to convey the combined intents of:
  - The exporting service, and
  - The intents that the exporting distribution provider adds, and
  - The intents that the importing distribution provider adds.
- Any additional properties listed in the Endpoint Description that should not be excluded. See Endpoint Description on page 344 for more details about the properties in the Endpoint Description.

A Remote Service Admin service must strictly follow the rules for importing a service as outlined in the Remote Services chapter.

The Remote Service Admin must return an ImportRegistration object or null. Even if an Import Registration is returned, it can still be an invalid registration, see Invalid Registrations on page 352 if the setup of the connection failed asynchronously. The Import Registration must always be a new object. Each valid Import Registration corresponds to a proxy service, potentially shared, that was created for the given Endpoint. The issues around proxying are described in Proxying on page 352.

For each successful creation of an import registration, the Remote Service Admin service must publish an IMPORT_REGISTRATION event, if there is an error it must publish an IMPORT_ERROR, see Events on page 357.
For more information see Registration Life Cycle on page 351.

The Import Registration provides access to an ImportReference object with the getImportReference(). This object has the following methods:

- getImportedEndpoint() - Provides the Endpoint Description for this imported service.
- getImportedService() - Provides the Service Reference for the service proxy.

The Import Registration will remain open as long as:

- The corresponding remote Endpoint remains available, and
- The Remote Service Admin service is still in use by the Topology Manager that created the Import Registration.

That is, the Import Registrations are not permanent, any persistence is in the realm of the Topology Manager. See Registration Life Cycle on page 351 for more details.

122.5.3 Reflection

The Remote Service Admin service provides the following methods to get the list of the current exported and imported services:

- getExportedServices() - List the Export References for services that are exported by this Remote Service Admin service as directed by any of the Topology Managers.
- getImportedEndpoints() - List the Import References for services that have been imported by this Remote Service Admin service as directed by any of the Topology Managers.

122.5.4 Registration Life Cycle

The registration life cycle of imported and exported services is non-trivial because:

- Multiple Export Registrations can use to the same Endpoint.
- Multiple Import Registrations can use to the same service proxy.

For example, Topology Manager A could create an Export Registration for service S to Endpoint E. Topology Manager B could attempt to create exactly the same Endpoint E for service S. However, an Endpoint occupies a unique address and it is often not possible to create multiple Endpoints for the same address. A Remote Service Admin service must therefore detect the case that multiple registrations share the same Endpoint between Topology Manager A’s registration and B’s registration. However, if Topology Manager B now closes its Export Registration then Topology Manager A still assumes the availability of the Endpoint. This scenario is depicted in Figure 122.4 on page 351. A similar example can be made for an Import Registration. In that case the same service proxy can be shared between multiple Import Registrations.

**Figure 122.4 Sharing Endpoints and proxies**

To simplify the implementation of Topology Managers, the Remote Service Admin must make this sharing of Endpoints and proxies between different registrations transparent. Both the Import and
Export Registrations must be unique objects for every registration call. That is, even if the same Topology Manager creates a registration for the same Endpoint/proxy then it must still receive a new registration. Though reference counting of the close operations could be used to detect when an Endpoint or proxy can be cleared, it would require that the Topology Manager exactly matches the close calls with the creation of the registrations. However, it is very hard to ensure that the close method is only called once in certain error and cleanup scenarios. The Remote Service Admin must therefore return unique objects for all registrations and manage the cleanup of proxies and Endpoints internally, even if the close method is called multiple times on a registration.

A Remote Service Admin service must use a Service Factory for its service object to maintain separation between Topology Managers. All registrations obtained through a Remote Service Admin service are life cycle bound to the Topology Manager that created it. That is, if a Topology Manager ungets its Remote Service Admin service, all registrations obtained through this service must automatically be closed. This model ensures that all registrations are properly closed if either the Remote Service Admin or the Topology Manager stops because in both cases the framework performs the unget automatically.

### 122.5.5 Invalid Registrations

The Remote Service Admin service is explicitly allowed to return invalid Import and Export Registrations. First, in a communications stack it can take time to discover that there are issues, allowing the registration to return before it has completed can potentially save time. Second, it allows the Topology Manager to discover problems with the configuration information. Without the invalid Export Registrations, the Topology Manager would have to scan the log or associate the Remote Service Admin Events with a specific import/export method call, something that can be difficult to do.

If the registration is invalid, the `getException()` method must return a `Throwable` object. If the registration has initialized correctly, this method will return `null`. The `getExportReference()` and `getImportReference()` methods must throw an `Illegal State Exception` when the registration is invalid.

A Remote Service Admin service is allowed to block for a reasonable amount of time when any of these methods is called, including the `getException` method, to finish initialization.

An invalid registration can be considered as never having been opened, it is therefore not necessary to close it; however, closing an invalid or closed registration must be a dummy operation and never throw an Exception. However, a failed registration must generate a corresponding error event.

### 122.5.6 Proxying

It is the responsibility of the Remote Service Admin service to properly proxy an imported service. This specification does not mandate the technique used to proxy an Endpoint as a service in the OSGi framework. The OSGi Remote Services specification allows a distribution provider to limit what it can proxy.

One of the primary aspects of a proxy is to ensure class space consistency between the exporting bundle and importing bundles. This can require the generation of a proxy-per-bundle to match the proper class spaces. It is the responsibility of the Remote Service Admin to ensure that no Class Cast Exceptions occur.

A common technique to achieve maximum class space compatibility is to use a Service Factory. A Service Factory provides the calling bundle when it first gets the service, making it straightforward to verify the package version of the interface that the calling bundle uses. Knowing the bundle that requests the service allows the creation of specialized proxies for each bundle. The interface class(es) for the proxy can then be loaded directly from the bundle, ensuring class compatibility. Interfaces should be loadable by the bundle otherwise that bundle can not use the interface in its code. If an interface cannot be loaded then it can be skipped. A dedicated class loader can then be created that has visibility to all these interfaces and is used to define the proxy class. This design ensures proper visibility and consistency. Implementations can optimize this model by sharing compatible class loaders between bundles.
The proxy will have to call arbitrary methods on arbitrary services. This has a large number of security implications, see Security on page 363.

122.6 Discovery

The topology of the distributed system is decided by the Topology Manager. However, in a distributed environment, the Topology Manager needs to discover Endpoints in other frameworks. There is a very large number of ways how a Topology Manager could learn about other Endpoints, ranging from static configuration, a centralized administration, all the way to fully dynamic discovery protocols like the Service Location Protocol (SLP) or JGroups. To support the required flexibility, this specification defines an Endpoint Listener service that allows the dissemination of Endpoint information. This service provides a symmetric solution because the problem is symmetric: it is used by a Topology Manager to announce changes in its local topology as well as find out about other Endpoint Descriptions. Where those other Endpoint Descriptions come from can vary widely. This design is depicted in Figure 122.5 on page 353.

Figure 122.5 Examples

The design of the Endpoint Listener allows a federated registry of Endpoint Descriptions. Any party that is interested in Endpoint Descriptions should register an Endpoint Listener service. This will signal that it is interested in topology information to any Endpoint Description Providers. Each Endpoint Listener service must be registered with a service property that holds a set of filter strings to indicate the scope of its interest. These filters must match an Endpoint Description before the corresponding Endpoint Listener service is notified of the availability of an Endpoint Description. Scoping is intended to limit the delivery of unnecessary Endpoint Descriptions as well as signal the need for specific Endpoints.

A Topology Manager has knowledge of its local Endpoints and is likely to be only interested in remote Endpoints. It can therefore set the scope to only match remote Endpoint Descriptions. See Framework UUID on page 347 for how to limit the scope to local or remote Endpoints. At the same time, a Topology manager should inform any locally registered Endpoint Listener services with Endpoints that it has created or deleted.

This architecture allows many different use cases. For example, a bundle could display a map of the topology by registering an Endpoint Listener with a scope for local Endpoints. Another example is the use of SLP to announce local Endpoints to a network and to discover remote Endpoints from other parties on this network.
An instance of this design is shown in Figure 122.6 on page 354. In this figure, there are 3 frameworks that collaborate through some discovery bundle. The Top framework has created an Endpoint and decides to notify all Endpoint Listeners registered in this framework that are scoped to this new Endpoint. Local bundle D has set its scope to all Endpoint Descriptions that originate from its local framework, it therefore receives the Endpoint Description from T. Bundle D then sends the Endpoint Description to all its peers on the network.

In the Quark framework, the manager bundle T has expressed an interest by setting its scope to a filter that matches the Endpoint Description from the Top framework. When the bundle D on the Quark framework receives the Endpoint Description from bundle D on the Top framework, it matches it against all local Endpoint Listener’s scope. In this case, the local manager bundle T matches and is given the Endpoint Description. The manager then uses the Remote Service Admin service to import the exported service described by the given Endpoint Description.

Figure 122.6  
 Endpoint Discovery Architecture. T=Topology Manager, D=Discovery

The previous description is just one of the possible usages of the Endpoint Listener. For example, the discovery bundles could communicate the scopes to their peers. These peers could then register an Endpoint Listener per peer, minimizing the network traffic because Endpoint Descriptions do not have to be broadcast to all peers.

Another alternative usage is described in Endpoint Description Extender Format on page 358. In this chapter the extender pattern is used to retrieve Endpoint Descriptions from resources in locally active bundles.

122.6.1 Scope and Filters

An Endpoint Listener service is registered with the ENDPOINT_LISTENER_SCOPE service property. This property, which is String+, must be set and must contain at least one filter. If there is not at least one filter, then that Endpoint Listener must not receive any Endpoint Descriptions.

Each filter in the scope is applied against the properties of the Endpoint Description until one succeeds. Only if one succeeds is the Endpoint informed about the existence of an Endpoint.

The Endpoint Description is designed to reflect the properties of the imported service, there is therefore a correspondence with the filters that are used by bundles that are listening for service registrations. The purpose of this design is to match the filter available through Listener Hook services, see On Demand on page 356.

However, the purpose of the filters is more generic than just this use case. It can also be used to specify the interest in local Endpoints or remote Endpoints. For example, Topology Managers are only
interested in remote Endpoints while discoverers are only interested in local Endpoints. It is easy to
discriminate between local and remote by filtering on the endpoint.framework.uuid property. End-
point Descriptions contain the Universally Unique ID (UUID) of the originating framework. This
UUID must be available from the local framework as well. See Framework UUID on page 347.

122.6.2 Endpoint Listener Interface

The EndpointListener interface has the following methods:

- `endpointAdded(EndpointDescription,String)` - Notify the Endpoint Listener of a new Endpoint
  Description. The second parameter is the filter that matched the Endpoint Description. Register-
  ing the same Endpoint multiple times counts as a single registration.
- `endpointRemoved(EndpointDescription,String)` - Notify the Endpoint Listener that the provided
  Endpoint Description is no longer available.

These methods must only be called if the Endpoint Listener service has a filter in its scope that
matches the Endpoint Description properties. The reason for the filter string in the methods is to
simplify and speed up matching an Endpoint Description to the cause of interest. For example, if the
Listener Hook is used to do on demand import of services, then the filter can be associated with the
Listener Info of the hook, see On Demand on page 356. If multiple filters in the scope match the
Endpoint Description than the first filter in the scope must be passed.

The Endpoint Listener interface is idempotent. Endpoint Description Providers must inform an End-
point Listener service that is registered of all their matching Endpoints. The only way to find out
about all available Endpoints is to register an Endpoint Listener that is then informed by all avail-
able Endpoint Description Providers of their known Endpoint Descriptions that match their scope.

122.6.3 Endpoint Listener Implementations

An Endpoint Listener service tracks the known Endpoints in its given scope. There are potentially a
large number of bundles involved in creating this federated registry of Endpoints. To ensure that no
Endpoint Descriptions are orphaned or unnecessarily missed, an Endpoint Listener implementation
must follow the following rules:

- **Registration** - An Endpoint Listener service is called with an
  `endpointAdded(EndpointDescription,String)` method for all known Endpoint Descriptions that
  the bundles in the local framework are aware of.
- **Tracking providers** - An Endpoint Listener must track the bundles that provide it with Endpoint
  Descriptions. If a bundle that provided Endpoint Descriptions is stopped, all Endpoint Descrip-
  tions that were provided by that bundle must be removed. This can be implemented straightfor-
  wardly with a Service Factory.
- **Scope modification** - An Endpoint Listener is allowed to modify the set of filters in its scope
  through a service property modification. This modification must result in new and/or existing
  Endpoint Descriptions to be added, however, existing Endpoints that are no longer in scope are
  not required to be explicitly removed by the their sources. It is the responsibility for the End-
  point Listener to remove these orphaned Endpoint Description from its view.

Endpoint Descriptions can be added from different sources and providers of Endpoint Descriptions
often use asynchronous and potentially unreliable communications. An implementation must
therefore handle the addition of multiple equal Endpoint Descriptions from different sources as
well as from the same source. Implementations must not count the number of registrations, a re-
move operation of an Endpoint Description is final for each source. That is, if source A added End-
point Description e, then it can only be removed by source A. However, if source A added e multiple
times, then it only needs to be removed once. Removals of Endpoint Descriptions that have not been
added (or were removed before) should be ignored.
The discovery of Endpoints is a fundamentally indeterministic process and implementations of Endpoint Listener services should realize that there are no guarantees that an added Endpoint Description is always describing a valid Endpoint.

### 122.6.4 Endpoint Description Providers

The Endpoint Listener service is based on an asynchronous, unreliable, best effort model because there are few guarantees in a distributed world. It is the task of an Endpoint Description Provider, for example a discovery bundle, to keep the Endpoint Listener services up to date of any Endpoint Descriptions the provider is aware of and that match the tracked service’s scope.

If an Endpoint Listener service is registered, a provider must add all matching Endpoint Descriptions that it is aware of and match the tracked Endpoint Listener’s scope. This can be done during registration or asynchronously later. For example, it is possible to use the filters in the scope to request remote systems for any Endpoint Descriptions that match those filters. For expediency reasons, the service registration event should not be delayed until those results return; it is therefore applicable to add these Endpoint Descriptions later when the returns from the remote systems finally arrive.

A tracked Endpoint Listener is allowed to modify its scope by setting new properties on its Service Registration. An Endpoint Description provider must process the new scope and add any newly matching Endpoint Descriptions. It is not necessary to remove any Endpoint Descriptions that were added before but no longer match the new scope. Removing those orphaned descriptions is the responsibility of the Endpoint Listener implementation.

It is not necessary to remove any registered Endpoint Descriptions when the Endpoint Listener is unregistered; also here it is the responsibility of the Endpoint Listener to do the proper cleanup.

### 122.6.5 On Demand

A common distribution policy is to import services that are being listened for by local bundles. For example, when a bundle opens a Service Tracker on the Log Service, a Topology Manager could be notified and attempt to find a Log Service in the local cluster and then import this service in the local Service Registry.

The OSGi framework provides service hooks for exactly this purpose. A Topology Manager can register a Listener Hook service and receive the information about bundles that have specified an interests in specific services.

For example, a bundle creates the following Service Tracker:

```java
ServiceTracker st = new ServiceTracker(context, LogService.class.getName() );
st.open();
```

This Service Tracker will register a Service Listener with the OSGi framework. This will cause the framework to add a ListenerInfo to any Listener Hook services. The getFilter method on a ListenerInfo object provides a filter that is directly applicable for the Endpoint Listener’s scope. In the previous example, this would be the filter:

```
(objectClass=org.osgi.service.log.LogService)
```

A Topology Manager could verify if this listener is satisfied. That is, if it has at least one service. If no such service could be found, it could then add this filter to its Endpoint Listener’s scope to detect remote implementations of this service. If such an Endpoint is detected, it could then request the import of this service through the Remote Service Admin service.
122.7 Events

The Remote Service Admin service must synchronously inform any Remote Service Admin Listener services of events as they happen. Client of the events should return quickly and not perform any but trivial processing in the same thread.

The following event types are defined:

- **EXPORT_ERROR** - An exported service has run into an unrecoverable error, although the Export Registration has not been closed yet. The event carries the Export Registration as well as the Exception that caused the problem, if present.
- **EXPORT_REGISTRATION** - The Remote Service Admin has registered a new Export Registration.
- **EXPORT_UNREGISTRATION** - An Export Registration has been closed, the service is no longer exported and the Endpoint is no longer active when this was the last registration for that service/Endpoint combination.
- **EXPORT_WARNING** - An exported service is experiencing problems but the Endpoint is still available.
- **IMPORT_ERROR** - An imported service has run into a fatal error and has been shut down. The Import Registration should be closed by the Topology Manager that created them.
- **IMPORT_REGISTRATION** - A new Import Registration was created for a potentially existing service/Endpoint combination.
- **IMPORT_UNREGISTRATION** - An Import Registration was closed, removing the proxy if this was the last registration.
- **IMPORT_WARNING** - An imported service is experiencing problems but can continue to function.

The following properties are available on the event:

- **getType()** - The type of the event.
- **getException()** - Any exception, if present.
- **getExportReference()** - An export reference, if applicable.
- **getImportReference()** - An import reference, if applicable.
- **getSource()** - The source of the event, the Remote Service Admin service.

122.7.1 Event Admin Mapping

All Remote Service Admin events must be posted, which is asynchronously, to the Event Admin service, if present, under the following topic:

`org/osgi/service/remoteserviceadmin/<type>`

Where `<type>` represents the type of the event, for example IMPORT_ERROR.

The Event Admin event must have the following properties:

- **bundle** - (Bundle) The Remote Service Admin bundle
- **bundle.symbolicname** - (String) The Bundle Symbolic Name of the Remote Service Admin bundle
- **bundle.signer** - (String[]) Signer of the Remote Service Admin bundle
- **cause** - The exception, if present.
- **endpoint.service.id** - (Long) Remote service id, if present
- **endpoint.framework.uuid** - (String) Remote service's Framework UUID, if present
- **endpoint.id** - (String) The id of the Endpoint, if present
The Endpoint Description Extender format is a possibility to deliver Endpoint Descriptions in bundles. This section defines an XML schema and how to locate XML definition resources that use this schema to define Endpoint Descriptions. The definition resource is a simple property based model that can define the same information as the properties on an imported service. If a bundle with the description is ready (ACTIVE or lazy activation and in the STARTING state), then this static description can be disseminated through the Endpoint Listeners that have specified an interest in this description. If the bundle is stopped, the corresponding Endpoints must be removed.

XML documents containing remote service descriptions must be specified by the Remote-Service header in the manifest. The structure of the Remote Service header is:

```
Remote-Service ::= header // See Common Header Syntax in Core
```

The value of the header is a comma separated list of paths. A path is:

- A directory if it ends with a solidus ('/') . A directory is scanned for '*.xml' files.
- A path with wildcards. Such a path can use the wildcards in its last component, as defined in the findEntries method.
- A complete path, not having wildcards not ending in a solidus ('/' ).

The Remote-Service header has no architected directives or attributes, unrecognized attributes and directives must be ignored.

A Remote-Service manifest header specified in a fragment must be ignored. However, XML documents referenced by a bundle’s Remote-Service manifest header can be contained in attached fragments. The required behavior for this is implemented in the findEntries method.

The extender must process each XML document specified in this header. If an XML document specified by the header cannot be located in the bundle and its attached fragments, the extender must log an error message with the Log Service, if present, and continue.

For example:

```
Remote-Service: lib/, remote/osgi/*.dsc, cnf/google.xml
```

This matches all resources in the lib directory matching '*.xml', all resources in the /remote/osgi directory that end with *.dsc, as well as the google.xml resource in the cnf directory.

The namespace of these XML resources must be:

```
http://www.osgi.org/xmlns/rsa/v1.0.0
```

This namespace describes a set of Endpoint Descriptions, where each Endpoint Description can provide a set of properties. The structure of this schema is:

```
endpoint-descriptions  ::= <endpoint-description>*
endpoint-description   ::= <property>*
property               ::= ( <array> | <list> | <set>| <xml> )?
```
array ::= <value> *
list ::= <value> *
set ::= <value> *
xml ::= <*> *

This structure is depicted in Figure 122.7 on page 359.

Figure 122.7  Endpoint Description XML Structure

The property element has the attributes listed in table Table 122.2.

Table 122.2  Property Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>The required name of the property. The type maps to the XML Schema xsd:string type.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>value-type</td>
<td>String</td>
<td>The optional type name of the property, the default is String. Any value in the value attribute or the value element when collections are used must be converted to the corresponding Java types. If the primitive form, for example byte, is specified for non-array types, then the value must be silently converted to the corresponding wrapper type.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>The value. Must be converted to the specified type if this is not the String type. The value attribute must not be used when the property element has a child element.</td>
</tr>
</tbody>
</table>

A property can have an array, list, set, or xml child element. If a child element is present then it is an error if the value attribute is defined. It is also an error if there is no child element and no value attribute.

The array, list, or set are multi-valued. That is, they contain 0 or more value elements. A value element contains text (a string) that must be converted to the given value-type or if not specified, left as is. Conversion must trim the leading and trailing white space characters as defined in the Character.isWhitespace method. No trimming must be done for strings. An array of primitive integers like int[] \{1,42,97\} can be encoded as follows:

```xml
<property name="integers" value-type="int">
  <array>
    <value>1</value>
    <value>42</value>
    <value>97</value>
  </array>
</property>
```

The xml element is used to convey XML from other namespaces, it is allowed to contain one foreign Xml root element, with any number of children, that will act as the root element of an XML document. This root element will be included in the corresponding property as a string. The XML element must be a valid XML document but not contain the XML processing instructions, the part between the `<?` and `?>`. The value-type of the property must be String or not set when an xml element is used, using another type is invalid.

The xml element can be used to embed configuration information, making the Endpoint Description self contained.
The following is an example of an endpoint-descriptions resource.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<endpoint-descriptions
 xmlns="http://www.osgi.org/xmlns/rsa/v1.0.0">
<endpoint-description>
 <property name="service.intents">
  <list>
   <value>SOAP</value>
   <value>HTTP</value>
  </list>
 </property>
 <property name="endpoint.id" value="http://ws.acme.com:9000/hello"/>
 <property name="objectClass" value="com.acme.Foo"/>
 <property name="endpoint.package.version.com.acme" value="4.2"/>
 <property name="service.imported.configs" value="com.acme"/>
 <property name="com.acme.ws.xml">
  <xml>
   <config xmlns="http://acme.com/defs">
    <port>1029</port>
    <host>www.acme.com</host>
   </config>
  </xml>
 </property>
</endpoint-description>
</endpoint-descriptions>
```

Besides being in a separate resource, the static configuration as described here could also be part of a larger XML file. In that case the parser must ignore elements not part of the http://www.osgi.org/xmlns/rsa/v1.0.0 namespace schema.

### 122.8.1 XML Schema

This namespace of the schema is:

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

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:rsa="http://www.osgi.org/xmlns/rsa/v1.0.0"
 targetNamespace="http://www.osgi.org/xmlns/rsa/v1.0.0"
 elementFormDefault="qualified" version="1.0.1">
 <annotation>
  <documentation xml:lang="en">
   This is the XML Schema for endpoint descriptions used by
   the Remote Service Admin Specification. Endpoint descriptions
   are used to describe remote services to a client in cases
   where a real live Discovery system isn't used. An extender,
   such as a local Discovery Service can look for service
   descriptions in installed bundles and inform the Topology
   Manager of these remote services. The Topology Manager can then
   instruct the Remote Service Admin to create client proxies for
   these services.
  </documentation>
 </annotation>

 <element name="endpoint-descriptions" type="rsa:Tendpoint-descriptions" />

 <complexType name="Tendpoint-descriptions">
  <sequence>
   <element name="endpoint-description" type="rsa:Tendpoint-description"
    minOccurs="1" maxOccurs="unbounded" />
  </sequence>
 </complexType>
```

It is non-deterministic, per W3C XML Schema 1.0:
http://www.w3.org/TR/xmlschema-1/#cos-nonambig to use
namespace="##any" below.

```xml
<any namespace="##other" minOccurs="0" maxOccurs="unbounded"
       processContents="lax" />
</sequence>
<anyAttribute processContents="lax" />
</complexType>

<complexType name="Tendpoint-description">
  <annotation>
    <documentation xml:lang="en">
      A Distribution Provider can register a proxy with the properties provided. Whether or not it is instructed to do so, is up to the Topology Manager. If any 'intents' properties are specified then the Distribution Provider should only register a proxy if it can support those intents.
    </documentation>
  </annotation>
  <sequence>
    <element name="property" type="rsa:Tproperty" minOccurs="1"
             maxOccurs="unbounded"/>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"
           processContents="lax" />
  </sequence>
  <anyAttribute processContents="lax" />
</complexType>

<complexType name="Tproperty" mixed="true">
  <sequence>
    <choice minOccurs="0" maxOccurs="1">
      <element name="array" type="rsa:Tmulti-value"/>
      <element name="list" type="rsa:Tmulti-value"/>
      <element name="set" type="rsa:Tmulti-value"/>
      <element name="xml" type="rsa:Txml"/>
    </choice>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"
           processContents="lax" />
  </sequence>
  <attribute name="name" type="string" use="required" />
  <attribute name="value" type="string" use="optional" />
  <attribute name="value-type" type="rsa:Tvalue-types" default="String" use="optional" />
  <anyAttribute processContents="lax" />
</complexType>

<complexType name="Tmulti-value">
  <sequence>
    <element name="value" minOccurs="0" maxOccurs="unbounded" type="rsa:Tvalue"/>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"
           processContents="lax" />
  </sequence>
  <anyAttribute processContents="lax" />
</complexType>

<complexType name="Tvalue" mixed="true">
  <sequence>
    <element name="xml" minOccurs="0" maxOccurs="1" type="rsa:Txml"/>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"
           processContents="lax" />
  </sequence>
  <anyAttribute processContents="lax" />
</complexType>

<!-- Specifies the data type of a property or of the elements in a multi-value property. Numerical and boolean values are trimmed before they are processed. Simple types are automatically boxed if needed. Only the array data type allows for simple type values. When specifying a simple type on any other type of property it will automatically be boxed. -->

<simpleType name="Tvalue-types">
  <restriction base="string">
    <enumeration value="String" />
    <enumeration value="long" />
    <enumeration value="Long" />
    <enumeration value="double" />
    <enumeration value="Double" />
  </restriction>
</simpleType>
122.9 Security

From a security point of view distribution is a significant threat. A Distribution Provider requires very significant capabilities to be able to proxy services. In many situations it will be required to grant the distribution provider All Permission. It is therefore highly recommended that Distribution Providers use trusted links and ensure that it is not possible to attack a system through the Remote Services Admin service and used discovery protocols.

122.9.1 Import and Export Registrations

Import and Export Registrations are capabilities. That is, they can only be obtained when the caller has the proper permissions but once obtained they are no longer checked. The caller should therefore be careful to share those objects with other bundles. Export and Import References are free to share.

122.9.2 Framework UUID Runtime Permission

The Remote Service Admin bundle can have the need for reading and writing the System property or the Framework UUID when this is not set by the deployer, the launcher, or the framework. This requires the following permission:

```java
PropertyPermission[
    "org.osgi.framework.uuid","read,write"
]```
122.9.3 **Endpoint Permission**

The Remote Service Admin implementation requires a large set of permissions because it must be able to distribute potentially any service. Giving these extensive capabilities to all Topology Managers would make it harder to developer general Topology Managers that implements specific scenarios. For this reason, this specification provides an Endpoint Permission.

When an Endpoint Permission must be verified, it must be created with an Endpoint Description as argument, like:

```java
sm.checkPermission( new EndpointPermission(anEndpoint,localUUID,READ));
```

The standard name and action constructor is used to define a permission. The name argument is a filter expression. The filter for an Endpoint Permission is applied to the properties of an Endpoint Description. The localUUID must map to the UUID of the framework of the caller of this constructor, see Framework UUID on page 347. This localUUID is used to allow a the permissions to use the <<LOCAL>> magic name in the permission filter to refer to the local framework.

The filter expression can use the following magic value:

- **<<LOCAL>>** - This value represents the framework UUID of the framework that this bundle belongs to. The following example restricts the visibility to descriptions of local Endpoints:

  ```java
  ALLOW {
    ...EndpointPermission
    "(endpoint.framework.uuid=<<LOCAL>>)"
    "READ" }
  ```

An Endpoint Permission that has the actions listed in the following table.

<table>
<thead>
<tr>
<th>Action</th>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPORT</td>
<td>importService(EndpointDescription)</td>
<td>Import an Endpoint</td>
</tr>
<tr>
<td>EXPORT</td>
<td>exportService(ServiceReference,Map)</td>
<td>Export a service</td>
</tr>
<tr>
<td>READ</td>
<td>getExportedServices()</td>
<td>See the presence of distributed services. The IMPORT and EXPORT action imply READ, Distribution of events to the Remote Service Admin Listener. The Remote Service Admin must verify that the listener's bundle has the proper permission. No events should be delivered that are not implied.</td>
</tr>
<tr>
<td></td>
<td>getimportedEndpoints()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>remoteAdminEvent(RemoteServiceAdminEvent)</td>
<td></td>
</tr>
</tbody>
</table>

122.10 **org.osgi.service.remoteserviceadmin**

Remote Service Admin 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.remoteserviceadmin; version="[1.0,2.0)"
```
Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.remoteserviceadmin; version="[1.0,1.1)"
```

### 122.10.1 Summary

- **EndpointDescription** - A description of an endpoint that provides sufficient information for a compatible distribution provider to create a connection to this endpoint. An Endpoint Description is easy to transfer between different systems because it is property based where the property keys are strings and the values are simple types.
- **EndpointListener** - A white board service that represents a listener for endpoints.
- **EndpointPermission** - A bundle’s authority to export, import or read an Endpoint.
- **ExportReference** - An Export Reference associates a service with a local endpoint.
- **ExportRegistration** - An Export Registration associates a service to a local endpoint.
- **ImportReference** - An Import Reference associates an active proxy service to a remote endpoint.
- **ImportRegistration** - An Import Registration associates an active proxy service to a remote endpoint.
- **RemoteConstants** - Provide the definition of the constants used in the Remote Service Admin specification.
- **RemoteServiceAdmin** - A Remote Service Admin manages the import and export of services.
- **RemoteServiceAdminEvent** - Provides the event information for a Remote Service Admin event.
- **RemoteServiceAdminListener** - A RemoteServiceAdminEvent listener is notified synchronously of any export or import registrations and unregistrations.

### 122.10.2 public class EndpointDescription

A description of an endpoint that provides sufficient information for a compatible distribution provider to create a connection to this endpoint. An Endpoint Description is easy to transfer between different systems because it is property based where the property keys are strings and the values are simple types. This allows it to be used as a communications device to convey available endpoint information to nodes in a network. An Endpoint Description reflects the perspective of an importer. That is, the property keys have been chosen to match filters that are created by client bundles that need a service. Therefore the map must not contain any service.exported.* property and must contain the corresponding service.imported.* ones. The service.intents property must contain the intents provided by the service itself combined with the intents added by the exporting distribution provider. Qualified intents appear fully expanded on this property.

**Concurrency** Immutable

#### 122.10.2.1 public EndpointDescription(Map<String,?> properties)

- **properties** The map from which to create the Endpoint Description. The keys in the map must be type String and, since the keys are case insensitive, there must be no duplicates with case variation.

- Create an Endpoint Description from a Map.

  - The endpoint.id, service.imported.configs and objectClass properties must be set.

  - **Throws** IllegalArgumentException – When the properties are not proper for an Endpoint Description.

#### 122.10.2.2 public EndpointDescription(ServiceReference<?> reference,Map<String,?> properties)

- **reference** A service reference that can be exported.

- **properties** Map of properties. This argument can be null. The keys in the map must be type String and, since the keys are case insensitive, there must be no duplicates with case variation.

- Create an Endpoint Description based on a Service Reference and a Map of properties. The properties in the map take precedence over the properties in the Service Reference.
This method will automatically set the endpoint.framework.uuid and endpoint.service.id properties based on the specified Service Reference as well as the service.imported property if they are not specified as properties.

The endpoint.id, service.imported.configs and objectClass properties must be set.

Throws IllegalArgumentException – When the properties are not proper for an Endpoint Description

122.10.2.3 public boolean equals(Object other)

other The EndpointDescription object to be compared.

- Compares this EndpointDescription object to another object.

An Endpoint Description is considered to be equal to another Endpoint Description if their ids are equal.

Returns true if object is a EndpointDescription and is equal to this object; false otherwise.

122.10.2.4 public List<String> getConfigurationTypes()

- Returns the configuration types. A distribution provider exports a service with an endpoint. This endpoint uses some kind of communications protocol with a set of configuration parameters. There are many different types but each endpoint is configured by only one configuration type. However, a distribution provider can be aware of different configuration types and provide synonyms to increase the change a receiving distribution provider can create a connection to this endpoint. This value of the configuration types is stored in the RemoteConstants.SERVICE_IMPORTED_CONFIGS service property.

Returns An unmodifiable list of the configuration types used for the associated endpoint and optionally synonyms.

122.10.2.5 public String getFrameworkUUID()

- Return the framework UUID for the remote service, if present. The value of the remote framework uuid is stored in the RemoteConstants.ENDPOINT_FRAMEWORK_UUID endpoint property.

Returns Remote Framework UUID, or null if this endpoint is not associated with an OSGi framework having a framework uuid.

122.10.2.6 public String getId()

- Returns the endpoint’s id. The id is an opaque id for an endpoint. No two different endpoints must have the same id. Two Endpoint Descriptions with the same id must represent the same endpoint. The value of the id is stored in the RemoteConstants.ENDPOINT_ID property.

Returns The id of the endpoint, never null. The returned value has leading and trailing whitespace removed.

122.10.2.7 public List<String> getIntents()

- Return the list of intents implemented by this endpoint. The intents are based on the service.intents on an imported service, except for any intents that are additionally provided by the importing distribution provider. All qualified intents must have been expanded. This value of the intents is stored in the RemoteConstants.SERVICE_INTENTS service property.

Returns An unmodifiable list of expanded intents that are provided by this endpoint.

122.10.2.8 public List<String> getInterfaces()

- Provide the list of interfaces implemented by the exported service. The value of the interfaces is derived from the objectClass property.

Returns An unmodifiable list of Java interface names implemented by this endpoint.
122.10.2.9 public Version getPackageVersion(String packageName)

packageName
The name of the package for which a version is requested.

Provide the version of the given package name. The version is encoded by prefixing the given package name with endpoint.package.version., and then using this as an endpoint property key. For example:

endpoint.package.version.com.acme

The value of this property is in String format and will be converted to a Version object by this method.

Returns The version of the specified package or Version.emptyVersion if the package has no version in this Endpoint Description.

Throws IllegalArgumentException– If the version property value is not String.

122.10.2.10 public Map<String, Object> getProperties()

Returns all endpoint properties.

Returns An unmodifiable map referring to the properties of this Endpoint Description.

122.10.2.11 public long getServiceld()

Returns the service id for the service exported through this endpoint. This is the service id under which the framework has registered the service. This field together with the Framework UUID is a globally unique id for a service. The value of the remote service id is stored in the RemoteConstants.ENDPOINT_SERVICE_ID endpoint property.

Returns Service id of a service or 0 if this Endpoint Description does not relate to an OSGi service.

122.10.2.12 public int hashCode()

Returns a hash code value for the object.

Returns An integer which is a hash code value for this object.

122.10.2.13 public boolean isSameService(EndpointDescription other)

other The Endpoint Description to look at

Answers if this Endpoint Description refers to the same service instance as the given Endpoint Description. Two Endpoint Descriptions point to the same service if they have the same id or their framework UUIDs and remote service ids are equal.

Returns True if this endpoint description points to the same service as the other

122.10.2.14 public boolean matches(String filter)

filter The filter to test.

Tests the properties of this EndpointDescription against the given filter using a case insensitive match.

Returns true If the properties of this EndpointDescription match the filter, false otherwise.

Throws IllegalArgumentException– If filter contains an invalid filter string that cannot be parsed.

122.10.2.15 public String toString()
public interface EndpointListener

A white board service that represents a listener for endpoints. An Endpoint Listener represents a participant in the distributed model that is interested in Endpoint Descriptions. This white board service can be used in many different scenarios. However, the primary use case is to allow a remote manager to be informed of Endpoint Descriptions available in the network and inform the network about available Endpoint Descriptions. Both the network bundle and the manager bundle register an Endpoint Listener service. The manager informs the network bundle about Endpoints that it creates. The network bundles then use a protocol like SLP to announce these local end-points to the network. If the network bundle discovers a new Endpoint through its discovery protocol, then it sends an Endpoint Description to all the Endpoint Listener services that are registered (except its own) that have specified an interest in that Endpoint. Endpoint Listener services can express their scope with the service property ENDPOINT_LISTENER_SCOPE. This service property is a list of filters. An Endpoint Description should only be given to a Endpoint Listener when there is at least one filter that matches the Endpoint Description properties. This filter model is quite flexible. For example, a discovery bundle is only interested in locally originating Endpoint Descriptions. The following filter ensure that it only sees local endpoints.

(org.osgi.framework.uuid=72dc5fd9-5f8f-4f8f-9821-9ebb433a5b72)

In the same vein, a manager that is only interested in remote Endpoint Descriptions can use a filter like:

(! (org.osgi.framework.uuid=72dc5fd9-5f8f-4f8f-9821-9ebb433a5b72))

Where in both cases, the given UUID is the UUID of the local framework that can be found in the Framework properties. The Endpoint Listener’s scope maps very well to the service hooks. A manager can just register all filters found from the Listener Hook as its scope. This will automatically provide it with all known endpoints that match the given scope, without having to inspect the filter string. In general, when an Endpoint Description is discovered, it should be dispatched to all registered Endpoint Listener services. If a new Endpoint Listener is registered, it should be informed about all currently known Endpoints that match its scope. If a getter of the Endpoint Listener service is unregistered, then all its registered Endpoint Description objects must be removed. The Endpoint Listener models a best effort approach. Participating bundles should do their utmost to keep the listeners up to date, but implementers should realize that many endpoints come through unreliable discovery processes.

Concurrency Thread-safe

public static final String ENDPOINT_LISTENER_SCOPE = "endpoint.listener.scope"

Specifies the interest of this listener with filters. This listener is only interested in Endpoint Descriptions where its properties match the given filter. The type of this property must be String+.

public void endpointAdded(EndpointDescription endpoint,String matchedFilter)

endpoint The Endpoint Description to be published
matchedFilter The filter from the ENDPOINT_LISTENER_SCOPE that matched the endpoint, must not be null.

Register an endpoint with this listener. If the endpoint matches one of the filters registered with the ENDPOINT_LISTENER_SCOPE service property then this filter should be given as the matchedFilter parameter. When this service is first registered or it is modified, it should receive all known endpoints matching the filter.

does not use

public void endpointRemoved(EndpointDescription endpoint,String matchedFilter)

endpoint The Endpoint Description that is no longer valid.
matchedFilter The filter from the ENDPOINT_LISTENER_SCOPE that matched the endpoint, must not be null.
Remove the registration of an endpoint. If an endpoint that was registered with the endpointAdded(EndpointDescription, String) method is no longer available then this method should be called. This will remove the endpoint from the listener. It is not necessary to remove endpoints when the service is unregistered or modified in such a way that not all endpoints match the interest filter anymore.

122.10.4  

**public final class EndpointPermission**  
**extends Permission**

A bundle's authority to export, import or read an Endpoint.

- The export action allows a bundle to export a service as an Endpoint.
- The import action allows a bundle to import a service from an Endpoint.
- The read action allows a bundle to read references to an Endpoint.

Permission to read an Endpoint is required in order to detect events regarding an Endpoint. Untrusted bundles should not be able to detect the presence of certain Endpoints unless they have the appropriate EndpointPermission to read the specific service.

**Concurrency**  
Thread-safe

122.10.4.1  
**public static final String EXPORT = "export"**

The action string export. The export action implies the read action.

122.10.4.2  
**public static final String IMPORT = "import"**

The action string import. The import action implies the read action.

122.10.4.3  
**public static final String READ = "read"**

The action string read.

122.10.4.4  
**public EndpointPermission(String filterString,String actions)**

- **filterString**  
  The filter string or "*" to match all endpoints.
- **actions**  
  The actions read, import, or export.

Create a new EndpointPermission with the specified filter.

The filter will be evaluated against the endpoint properties of a requested EndpointPermission.

There are three possible actions: read, import and export. The read action allows the owner of this permission to see the presence of distributed services. The import action allows the owner of this permission to import an endpoint. The export action allows the owner of this permission to export a service.

**Throws**  
IllegalArgumentException—If the filter has an invalid syntax or the actions are not valid.

122.10.4.5  
**public EndpointPermission(EndpointDescription endpoint,String localFrameworkUUID,String actions)**

- **endpoint**  
  The requested endpoint.
- **localFrameworkUUID**  
  The UUID of the local framework. This is used to support matching the endpoint.framework.uuid endpoint property to the \(<LOCAL>\) value in the filter expression.
- **actions**  
  The actions read, import, or export.

Creates a new requested EndpointPermission object to be used by code that must perform checkPermission. EndpointPermission objects created with this constructor cannot be added to an EndpointPermission permission collection.

**Throws**  
IllegalArgumentException—If the endpoint is null or the actions are not valid.
122.10.4.6  public boolean equals(Object obj)

obj  The object to test for equality.

□  Determines the equality of two EndpointPermission objects. Checks that specified object has the same name, actions and endpoint as this EndpointPermission.

Returns  true if obj is a EndpointPermission, and has the same name, actions and endpoint as this EndpointPermission object; false otherwise.

122.10.4.7  public String getActions()

□  Returns the canonical string representation of the actions. Always returns present actions in the following canonical order: read, import, export.

Returns  The canonical string representation of the actions.

122.10.4.8  public int hashCode()

□  Returns the hash code value for this object.

Returns  Hash code value for this object.

122.10.4.9  public boolean implies(Permission p)

p  The target permission to check.

□  Determines if a EndpointPermission object "implies" the specified permission.

Returns  true if the specified permission is implied by this object; false otherwise.

122.10.4.10  public PermissionCollection newPermissionCollection()

□  Returns a new PermissionCollection object for storing EndpointPermission objects.

Returns  A new PermissionCollection object suitable for storing EndpointPermission objects.

122.10.5  public interface ExportReference

An Export Reference associates a service with a local endpoint. The Export Reference can be used to reference an exported service. When the service is no longer exported, all methods must return null.

Concurrency  Thread-safe

Provider Type  Consumers of this API must not implement this type

122.10.5.1  public EndpointDescription getExportedEndpoint()

□  Return the Endpoint Description for the local endpoint.

Returns  The Endpoint Description for the local endpoint. Must be null when the service is no longer exported.

122.10.5.2  public ServiceReference<?> getExportedService()

□  Return the service being exported.

Returns  The service being exported. Must be null when the service is no longer exported.

122.10.6  public interface ExportRegistration

An Export Registration associates a service to a local endpoint. The Export Registration can be used to delete the endpoint associated with this registration. It is created with the RemoteServiceAdmin.exportService(ServiceReference,Map) method. When this Export Registration has been closed, all methods must return null.
122.10.6.1 public void close()

    □ Delete the local endpoint and disconnect any remote distribution providers. After this method returns, all methods must return null. This method has no effect when this registration has already been closed or is being closed.

122.10.6.2 public Throwable getException()

    □ Return the exception for any error during the export process. If the Remote Service Admin for some reason is unable to properly initialize this registration, then it must return an exception from this method. If no error occurred, this method must return null. The error must be set before this Export Registration is returned. Asynchronously occurring errors must be reported to the log.

    Returns The exception that occurred during the initialization of this registration or null if no exception occurred.

122.10.6.3 public ExportReference getExportReference()

    □ Return the Export Reference for the exported service.

    Returns The Export Reference for this registration.

    Throws IllegalStateException – When this registration was not properly initialized. See getException().

122.10.7 public interface ImportReference

An Import Reference associates an active proxy service to a remote endpoint. The Import Reference can be used to reference an imported service. When the service is no longer imported, all methods must return null.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

122.10.7.1 public EndpointDescription getImportedEndpoint()

    □ Return the Endpoint Description for the remote endpoint.

    Returns The Endpoint Description for the remote endpoint. Must be null when the service is no longer imported.

122.10.7.2 public ServiceReference<?> getImportedService()

    □ Return the Service Reference for the proxy for the endpoint.

    Returns The Service Reference to the proxy for the endpoint. Must be null when the service is no longer imported.

122.10.8 public interface ImportRegistration

An Import Registration associates an active proxy service to a remote endpoint. The Import Registration can be used to delete the proxy associated with an endpoint. It is created with the RemoteServiceAdmin.importService(EndpointDescription) method. When this Import Registration has been closed, all methods must return null.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type
122.10.8.1 public void close()

☐ Close this Import Registration. This must close the connection to the endpoint and unregister the proxy. After this method returns, all other methods must return null. This method has no effect when this registration has already been closed or is being closed.

122.10.8.2 public Throwable getException()

☐ Return the exception for any error during the import process. If the Remote Service Admin for some reasons is unable to properly initialize this registration, then it must return an exception from this method. If no error occurred, this method must return null. The error must be set before this Import Registration is returned. Asynchronously occurring errors must be reported to the log.

Returns The exception that occurred during the initialization of this registration or null if no exception occurred.

122.10.8.3 public ImportReference getImportReference()

☐ Return the Import Reference for the imported service.

Returns The Import Reference for this registration.

Throws IllegalStateException – When this registration was not properly initialized. See getException().

122.10.9 public class RemoteConstants

Provide the definition of the constants used in the Remote Service Admin specification.

Concurrency Immutable

122.10.9.1 public static final String ENDPOINT_FRAMEWORK_UUID = "endpoint.framework.uuid"

Endpoint property identifying the universally unique id of the exporting framework. Can be absent if the corresponding endpoint is not for an OSGi service.

The value of this property must be of type String.

122.10.9.2 public static final String ENDPOINT_ID = "endpoint.id"

Endpoint property identifying the id for this endpoint. This service property must always be set.

The value of this property must be of type String.

122.10.9.3 public static final String ENDPOINT_PACKAGE_VERSION_ = "endpoint.package.version."

Prefix for an endpoint property identifying the interface Java package version for an interface. For example, the property endpoint.package.version.com.acme=1.3 describes the version of the package for the com.acme.Foo interface. This endpoint property for an interface package does not have to be set. If not set, the value must be assumed to be 0.

Since endpoint properties are stored in a case insensitive map, case variants of a package name are folded together.

The value of properties having this prefix must be of type String.

122.10.9.4 public static final String ENDPOINT_SERVICE_ID = "endpoint.service.id"

Endpoint property identifying the service id of the exported service. Can be absent or 0 if the corresponding endpoint is not for an OSGi service.

The value of this property must be of type Long.

122.10.9.5 public static final String REMOTE_CONFIGS_SUPPORTED = "remote.configs.supported"

Service property identifying the configuration types supported by a distribution provider. Registered by the distribution provider on one of its services to indicate the supported configuration types.
The value of this property must be of type String, String[], or Collection of String.

See Also Remote Services Specification

122.10.9.6 public static final String REMOTE_INTENTS_SUPPORTED = "remote.intents.supported"

Service property identifying the intents supported by a distribution provider. Registered by the distribution provider on one of its services to indicate the vocabulary of implemented intents.

The value of this property must be of type String, String[], or Collection of String.

See Also Remote Services Specification

122.10.9.7 public static final String SERVICE_EXPORTED_CONFIGS = "service.exported.configs"

Service property identifying the configuration types that should be used to export the service. Each configuration type represents the configuration parameters for an endpoint. A distribution provider should create an endpoint for each configuration type that it supports.

This property may be supplied in the properties Dictionary object passed to the BundleContext.registerService method. The value of this property must be of type String, String[], or Collection of String.

See Also Remote Services Specification

122.10.9.8 public static final String SERVICE_EXPORTED_INTENTS = "service.exported.intents"

Service property identifying the intents that the distribution provider must implement to distribute the service. Intents listed in this property are reserved for intents that are critical for the code to function correctly, for example, ordering of messages. These intents should not be configurable.

This property may be supplied in the properties Dictionary object passed to the BundleContext.registerService method. The value of this property must be of type String, String[], or Collection of String.

See Also Remote Services Specification

122.10.9.9 public static final String SERVICE_EXPORTED_INTENTS_EXTRA = "service.exported.intents.extra"

Service property identifying the extra intents that the distribution provider must implement to distribute the service. This property is merged with the service.exported.intents property before the distribution provider interprets the listed intents; it has therefore the same semantics but the property should be configurable so the administrator can choose the intents based on the topology. Bundles should therefore make this property configurable, for example through the Configuration Admin service.

This property may be supplied in the properties Dictionary object passed to the BundleContext.registerService method. The value of this property must be of type String, String[], or Collection of String.

See Also Remote Services Specification

122.10.9.10 public static final String SERVICE_EXPORTED_INTERFACES = "service.exported.interfaces"

Service property marking the service for export. It defines the interfaces under which this service can be exported. This list must be a subset of the types under which the service was registered. The single value of an asterisk (\u002A) indicates all the interface types under which the service was registered excluding the non-interface types. It is strongly recommended to only export interface types and not concrete classes due to the complexity of creating proxies for some type of concrete classes.

This property may be supplied in the properties Dictionary object passed to the BundleContext.registerService method. The value of this property must be of type String, String[], or Collection of String.
122.10.9.11 public static final String SERVICE_IMPORTED = "service.imported"

Service property identifying the service as imported. This service property must be set by a distribution provider to any value when it registers the endpoint proxy as an imported service. A bundle can use this property to filter out imported services.

The value of this property may be of any type.

See Also Remote Services Specification

122.10.9.12 public static final String SERVICE_IMPORTED_CONFIGS = "service.imported.configs"

Service property identifying the configuration types used to import the service. Any associated properties for this configuration types must be properly mapped to the importing system. For example, a URL in these properties must point to a valid resource when used in the importing framework. If multiple configuration types are listed in this property, then they must be synonyms for exactly the same remote endpoint that is used to export this service.

The value of this property must be of type String, String[], or Collection of String.

See Also Remote Services Specification, SERVICE_IMPORTED_CONFIGS

122.10.9.13 public static final String SERVICE_INTENTS = "service.intents"

Service property identifying the intents that this service implement. This property has a dual purpose:

- A bundle can use this service property to notify the distribution provider that these intents are already implemented by the exported service object.
- A distribution provider must use this property to convey the combined intents of: The exporting service, and the intents that the exporting distribution provider adds, and the intents that the importing distribution provider adds.

To export a service, a distribution provider must expand any qualified intents. Both the exporting and importing distribution providers must recognize all intents before a service can be distributed.

The value of this property must be of type String, String[], or Collection of String.

See Also Remote Services Specification

122.10.10 public interface RemoteServiceAdmin

A Remote Service Admin manages the import and export of services. A Distribution Provider can expose a control interface. This interface allows a Topology Manager to control the export and import of services. The API allows a Topology Manager to export a service, to import a service, and find out about the current imports and exports.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

122.10.10.1 public Collection<ExportRegistration> exportService(ServiceReference<?> reference,Map<String,?> properties)

reference The Service Reference to export.

properties The properties to create a local Endpoint that can be implemented by this Remote Service Admin. If this is null, the Endpoint will be determined by the properties on the service. The properties are the same as given for an exported service. They override any properties in the specified Service Reference (case insensitive). The properties objectClass and service.id, in any case variant, are ignored. Those properties in the Service Reference cannot be overridden. This parameter can be null, this should be treated as an empty map.
Export a service to a given Endpoint. The Remote Service Admin must create an Endpoint from the given description that can be used by other Distribution Providers to connect to this Remote Service Admin and use the exported service. The property keys of a Service Reference are case insensitive while the property keys of the specified properties map are case sensitive. A property key in the specified properties map must therefore override any case variant property key in the properties of the specified Service Reference.

If the caller does not have the appropriate EndpointPermission[endpoint,EXPORT] for an Endpoint, and the Java Runtime Environment supports permissions, then the getException method on the corresponding returned ExportRegistration will return a SecurityException.

Returns A Collection of ExportRegistrations for the specified Service Reference and properties. Multiple Export Registrations may be returned because a single service can be exported to multiple Endpoints depending on the available configuration type properties. The result is never null but may be empty if this Remove Service Admin does not recognize any of the configuration types.

Throws IllegalArgumentException – If any of the properties has a value that is not syntactically correct or if the service properties and the overlaid properties do not contain a RemoteConstants.SERVICE_EXPORTED_INTERFACES entry.

UnsupportedOperationException – If any of the intents expressed through the properties is not supported by the distribution provider.

122.10.10.2 public Collection<ExportReference> getExportedServices()

Return the currently active Export References.

If the caller does not have the appropriate EndpointPermission[endpoint,READ] for an Endpoint, and the Java Runtime Environment supports permissions, then returned collection will not contain a reference to the exported Endpoint.

Returns A Collection of ExportReferences that are currently active.

122.10.10.3 public Collection<ImportReference> getImportedEndpoints()

Return the currently active Import References.

If the caller does not have the appropriate EndpointPermission[endpoint,READ] for an Endpoint, and the Java Runtime Environment supports permissions, then returned collection will not contain a reference to the imported Endpoint.

Returns A Collection of ImportReferences that are currently active.

122.10.10.4 public ImportRegistration importService(EndpointDescription endpoint)

endpoint The Endpoint Description to be used for import.

Return a service from an Endpoint. The Remote Service Admin must use the given Endpoint to create a proxy. This method can return null if the service could not be imported.

Returns An Import Registration that combines the Endpoint Description and the Service Reference or null if the Endpoint could not be imported.

Throws SecurityException – If the caller does not have the appropriate EndpointPermission[endpoint,IMPORT] for the Endpoint, and the Java Runtime Environment supports permissions.

122.10.11 public class RemoteServiceAdminEvent

Provides the event information for a Remote Service Admin event.

Concurrency Immutable
122.10.11.1  public static final int EXPORT_ERROR = 6
A fatal exporting error occurred. The Export Registration has been closed.

122.10.11.2  public static final int EXPORT_REGISTRATION = 2
Add an export registration. The Remote Service Admin will call this method when it exports a service. When this service is registered, the Remote Service Admin must notify the listener of all existing Export Registrations.

122.10.11.3  public static final int EXPORT_UNREGISTRATION = 3
Remove an export registration. The Remote Service Admin will call this method when it removes the export of a service.

122.10.11.4  public static final int EXPORT_WARNING = 7
A problematic situation occurred, the export is still active.

122.10.11.5  public static final int IMPORT_ERROR = 5
A fatal importing error occurred. The Import Registration has been closed.

122.10.11.6  public static final int IMPORT_REGISTRATION = 1
Add an import registration. The Remote Service Admin will call this method when it imports a service. When this service is registered, the Remote Service Admin must notify the listener of all existing Import Registrations.

122.10.11.7  public static final int IMPORT_UNREGISTRATION = 4
Remove an import registration. The Remote Service Admin will call this method when it removes the import of a service.

122.10.11.8  public static final int IMPORT_WARNING = 8
A problematic situation occurred, the import is still active.

122.10.11.9  public RemoteServiceAdminEvent(int type, Bundle source, ExportReference exportReference, Throwable exception)
type  The event type.
source  The source bundle, must not be null.
exportReference  The exportReference, can not be null.
exception  Any exceptions encountered, can be null.
□ Create a Remote Service Admin Event for an export notification.

122.10.11.10 public RemoteServiceAdminEvent(int type, Bundle source, ImportReference importReference, Throwable exception)
type  The event type.
source  The source bundle, must not be null.
importReference  The importReference, can not be null.
exception  Any exceptions encountered, can be null.
□ Create a Remote Service Admin Event for an import notification.

122.10.11.11 public Throwable getException()
□ Return the exception for this event.
Returns The exception or null.

122.10.11.12 public ExportReference getExportReference()
☐ Return the Export Reference for this event.
Returns The Export Reference or null.

122.10.11.13 public ImportReference getImportReference()
☐ Return the Import Reference for this event.
Returns The Import Reference or null.

122.10.11.14 public Bundle getSource()
☐ Return the bundle source of this event.
Returns The bundle source of this event.

122.10.11.15 public int getType()
☐ Return the type of this event.
Returns The type of this event.

122.10.12 public interface RemoteServiceAdminListener

A RemoteServiceAdminEvent listener is notified synchronously of any export or import registrations and unregistrations.

If the Java Runtime Environment supports permissions, then filtering is done. RemoteServiceAdminEvent objects are only delivered to the listener if the bundle which defines the listener object's class has the appropriate EndpointPermission[endpoint,READ] for the endpoint referenced by the event.

See Also RemoteServiceAdminEvent
Concurrency Thread-safe

122.10.12.1 public void remoteAdminEvent(RemoteServiceAdminEvent event)

event The RemoteServiceAdminEvent object.
☐ Receive notification of any export or import registrations and unregistrations as well as errors and warnings.

122.11 References

[1] OSGi Service Property Namespace
http://www.osgi.org Specifications/ServicePropertyNamespace
[2] UUIDs
http://en.wikipedia.org/wiki/Universally_Unique_Identifier
[3] Service Location Protocol (SLP)
[4] JGroups
http://www.jgroups.org/
[5] UDDI
http://en.wikipedia.org/wiki/Universal_Description_Discovery_and_Integration
123 JTA Transaction Services Specification

Version 1.0

123.1 Introduction

Transactions are the key abstraction to provide reliability with large scale distributed systems and are a primary component of enterprise systems. This specification provides an OSGi service based design for the Java Transaction Architecture (JTA) Specification, which describes the standard transaction model for Java applications. Providing the JTA specification as a service based model enables the use of independent implementations. This JTA Transaction Services Specification provides a managed model, where an Application Container (such as the Java EE EJB container) manages the transaction and the enlistment of resources, and an unmanaged model, where each application is responsible for these tasks itself.

This specification provides a brief overview of JTA and then the use of it through 3 transaction services: User Transaction, Transaction Manager, and Transaction Synchronization.

This specification is based on [1] Java Transaction API Specification 1.1.

123.1.1 Essentials

- Portability - It is important that applications are easy to port from other environments that support JTA.
- Pluggability - Allow different vendors to provide implementations of this specification.
- JTA Compatible - Support full JTA 1.1 Specification

123.1.2 Entities

- JTA Provider - Implementation of this specification. It is responsible, on request from a Transaction Originator, for starting and ending transactions and coordinating the work of Resource Managers that become involved in each Transaction. This entity provides the User Transaction service, Transaction Manager service, and the Transaction Synchronization Registry service.
- Transaction - An atomic unit of work that is associated with a thread of execution.
- Transaction Originator - An Application or its Container, that directs the JTA Provider to begin and end Transactions.
- User Transaction - A service used by a Transaction Originator for beginning and ending transactions.
- Transaction Manager - A service used by a Transaction Originator for managing both transaction demarcation and enlistment of Durable Resources or Volatile Resources.
- Transaction Synchronization Registry - A service for enlistment of Volatile Resources for getting notifications before and after ending Transactions.
- Application Bundle - An entity that initiates work that executes under a Transaction.
- Container - An entity that is distinct from the Application and which provides a managed environment for Applications. Unmanaged environments do not distinguish between the Application and Container entities.
• **Resource Manager** - Provides the transnational resources whose work is externally coordinat-ed by a JTA Provider. Examples of Resource Managers include databases, Java Message Service providers and enterprise information systems.

• **Durable Resource** - A resource whose work is made durable when the Transaction is successfully committed. Durable Resources can be enlisted with a Transaction to ensure that work is performed within the scope of the Transaction and to participate in the outcome of a Transaction. Durable Resource enlistment is the responsibility of the Application Bundle or its Container. Durable Resources implement the `javax.transaction.xa.XAResource` interface.

• **Volatile Resource** - Resources that are associated with a Transaction but are no longer needed after the Transaction, for example transaction-scoped caches. Volatile Resources are registered with the JTA Provider to receive notifications before and after the outcome of the Transaction. Volatile Resources implement the `javax.transaction.Synchronization` interface.

• **Transaction Services** - The triplet of the User Transaction, Transaction Manager, and Transaction Synchronization Registry services registered by the JTA Provider.

**Figure 123.1 Transaction Service Specification Entities**

```
<table>
<thead>
<tr>
<th>JTA Provider</th>
<th>XA Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Transaction</td>
<td></td>
</tr>
<tr>
<td>Transaction Manager</td>
<td></td>
</tr>
<tr>
<td>Transaction Synchronization Registry</td>
<td></td>
</tr>
<tr>
<td>Managed Application Impl</td>
<td></td>
</tr>
<tr>
<td>Application Container Impl</td>
<td></td>
</tr>
<tr>
<td>Resource Manager Impl</td>
<td></td>
</tr>
<tr>
<td>XA Resource Impl</td>
<td></td>
</tr>
</tbody>
</table>
```

**123.1.3 Dependencies**

This specification is based on the following packages:

- `javax.transaction`
- `javax.transaction.xa`

These packages must be exported as version 1.1.

**123.1.4 Synopsis**

The JTA Provider register the Transaction Services:

- **User Transaction** - Offers transaction demarcation capabilities to an Application bundle.
- **Transaction Manager** - Offers transaction demarcation and further transaction management capabilities to an Application Bundle or an Application Container.
- **Transaction Synchronization Registry** - Offers a callback registration service for volatile transactional participants wishing to be notified of the completion of the transaction.

A JTA Provider must register these services when it is started. A JTA Provider may put restrictions on which bundles can use these services. For example, in a Java EE environment, the JTA Provider does not expose the `TransactionManager` interface to applications. An OSGi environment which
supports the Java EE specifications will typically provide access to the Transaction Manager service only to Java EE Containers.

A typical example of the use of a transaction is for transferring money from one bank account to another. Two Durable Resources are involved, one provided by the database from which the money is to be withdrawn and another provided by the database to which the money will be deposited. An Application Bundle acting as the Transaction Originator gets the User Transaction service and uses it to begin a transaction. This transaction is associated with the current thread (implicitly) by the JTA Provider. On the same thread of execution, the Application Bundle connects to the database from which the money is to be withdrawn and updates the balance in the source account by the amount to be debited.

The database is a resource manager whose connections have associated XA Resources; the first time a connection is used within the scope of a new transaction the Application Bundle, or a Container, obtains the XA Resource associated with the connection and enlists it with the JTA Provider through the Transaction Manager service. On the same thread of execution, the Application Bundle connects to the second database and updates the balance in the target account by the amount to be credited. An XA Resource for the second connection is enlisted with the Transaction Manager service as well by the Application Bundle or a Container.

Now that the money has been transferred the Transaction Originator requests a commit of the Transaction (on the same thread of execution) via the User Transaction Service, causing the JTA Provider to initiate the two-phase commit process with the two Resource Managers through the enlisted XA Resources. The transaction is then atomically committed or rolled back.

### 123.2 JTA Overview

A transaction is a unit of work in which interactions with multiple participants can be coordinated by a third party such that the final outcome of these interactions has well-defined transactional semantics. A variety of well-known transaction models exist with specific characteristics; the transactions described in this specification provide Atomic Consistent Isolated and Durable (ACID) semantics as defined in [2], XA+ Specification whereby all the participants in a transaction are coordinated to an atomic outcome in which the work of all the participants is either completely committed or completely rolled back.

The [2], XA+ Specification defines a Distributed Transaction Processing (DTP) software architecture for transactional work that is distributed across multiple Resource Managers and coordinated externally by a Transaction Manager using the two-phase commit XA protocol. The DTP architecture defines the roles of the Transaction Manager and Resource Manager; this specification uses the term JTA Provider rather than Transaction Manager to distinguish it from the Transaction Manager service. Note that Distributed Transaction Processing does not imply distribution of transactions across multiple frameworks or JVMs.

The [1], Java Transaction API Specification 1.1 defines the Java interfaces required for the management of transactions on the enterprise Java platform.

### 123.2.1 Global and Local Transactions

A transaction may be a local transaction or a global transaction. A local transaction is a unit of work that is local to a single Resource Manager and may succeed or fail independently of the work of other Resource Managers. A global transaction, sometimes referred to as a distributed transaction, is a unit of work that may encompass multiple Resource Managers and is coordinated by a JTA Provider external to the Resource Manager(s) as described in the DTP architecture. The term transaction in this specification always refers to a global transaction.

The JTA Provider is responsible for servicing requests from a Transaction Originator to create and complete transactions, it manages the state of each transaction it creates, the association of each
transaction with the thread of execution, and the coordination of any Resource Managers that be-
come involved in the global transaction. The JTA Provider ensures that each transaction is associat-
ed with, at most, one application thread at a time and provides the means to move that association
from one thread to another as needed.

The model for resource commit coordination is the two phase commit XA protocol, with Resource
Managers being directed by the JTA Provider. The first time an Application accesses a Resource Man-
ger within the scope of a new global transaction, the Application, or its Container, obtains an XA
Resource from the Resource Manager and enlists this XA Resource with the JTA Provider.

At the end of a transaction, the Transaction Originator must decide whether to initiate a commit or
rollback request for all the changes made within the scope of the Transaction. The Transaction Orig-
inator requests that the JTA Provider completes the transaction. The JTA Provider then negotiates
with each enlisted Resource Manager to reach a coordinated outcome. A failure in the transaction at
any point before the second phase of two-phase commit results in the transaction being rolled back.
XA is a presumed abort protocol and implementations of XA-compliant JTA Providers and Resource
Managers can be highly optimized to perform no logging of transactional state until a commit de-
cision is required. A Resource Manager durably records its prepare decision, and a JTA Provider
durably records any commit decision it makes. Failures between a decision on the outcome of a
transaction and the enactment of that outcome are handled during transaction recovery to ensure the
atomic outcome of the transaction.

123.2.2 Durable Resource

Durable Resources are provided by Resource Managers and must implement the XAResource inter-
face described in the [1] Java Transaction API Specification 1.1. An XAResource object is enlisted with
a transaction to ensure that the work of the Resource Manager is associated with the correct transac-
tion and to participate in the two-phase commit process. The XAResource interface is driven by the
JTA Provider during the completion of the transaction and is used to direct the Resource Manager to
commit or rollback any changes made under the corresponding transaction.

123.2.3 Volatile Resource

Volatile resources are components that do not participate in the two phase commit but are called
immediately prior to and after the two phase commit. They implement the [1] Java Transaction API
Specification 1.1 Synchronization interface. If a request is made to commit a transaction then the
volatile participants have the opportunity to perform some before completion processing such as
flushing cached updates to persistent storage. Failures during the before completion processing must
cause the transaction to rollback. In both the commit and rollback cases the volatile resources are
called after two phase commit to perform after completion processing. After completion procession
cannot affect the outcome of the transaction.

123.2.4 Threading

As noted above in Global and Local Transactions on page 381, a global transaction must not be asso-
ciated with more than one application thread at a time but can be moved over time from one applica-
tion thread to another. In some environments Applications run in containers which restrict the
ability of the Application component to explicitly manage the transaction-thread association by re-
stricting access to the Transaction Manager. For example, Java EE application servers provide web
and EJB Containers for application components and, while the Containers themselves can explicitly
manage transaction-thread associations, these containers do not allow the Applications to do so. Ap-
plications running in these containers are required to complete any transactions they start on that
same application thread. In general, Applications that run inside a Container must follow the rules
defined by that Container. For further details of the considerations specific to Java EE containers, see
123.3 Application

An Application is a bundle that may use transactions, either as a Transaction Originator or as a bundle that is called as part of an existing transaction. A Transaction Originator Application bundle starts a transaction and end it with a commit or rollback using the User Transaction or Transaction Manager service.

A Transaction Originator Application bundle may not make use of Resource Managers itself but may simply provide transaction demarcation and then call other bundles which do use Resource Managers. In such a case the Transaction Originator Application bundle requires only the use of the User Transaction service for transaction demarcation. The called bundles may use the Transaction Manager service if they use Resource Managers.

Application Bundles that use Resource Managers have to know the enlistment strategy for the Resource Managers they use. There are two possibilities:

- **Application Bundle Enlistment** - The Application Bundle must enlist the Resource Managers itself. For each Resource Manager it uses it must enlist that Resource Manager with the Transaction Manager.
- **Container-Managed Enlistment** - An Application runs in a container, such as a Java EE Container, which manages the Resource Manager enlistment on behalf of the Application.

These scenarios are explained in the following sections.

123.3.1 No Enlistment

A Transaction Originator Application bundle that uses no Resource Managers itself but starts a Transaction before calling another bundle may use the User Transaction service to control the Transaction demarcation.

For example, an Application can use the User Transaction service to begin a global transaction:

```java
UserTransaction ut = getUserTransaction();
ut.begin();
```

The User Transaction service associates a transaction with the current thread until that transaction is completed via:

```java
UserTransaction ut = getUserTransaction();
ut.commit();
```

Or the equivalent rollback method. The `getUserTransaction` method implementation (not shown) can get the User Transaction service directly from the service registry or from an injected field.

123.3.2 Application Bundle Enlistment

An Application Bundle is responsible for enlisting Resource Managers itself. That is, it must enlist Resource Manager it uses with the Transaction Manager service. The Transaction Manager service is an implementation of the JTA TransactionManager interface, registered by the JTA Provider.

For example, an Application Bundle can get an XADataSource object from a Data Source Factory service. Such a Data Source object can provide an XAConnection object that then can provide an XAResource object. XAResource objects can then be enlisted with the Transaction Manager service.

For example:

```java
TransactionManager tm;
XADataSource       left;
```
void acid() throws Exception {
    tm.begin();
    Transaction transaction = tm.getTransaction();
    try {
        XAConnection left = this.left.getXAConnection();
        XAConnection right = this.right.getXAConnection();
        transaction.enlistResource(left.getXAResource());
        transaction.enlistResource(right.getXAResource());
        doWork(left.getConnection(), right.getConnection());
        tm.commit();
    } catch (Throwable t) {
        tm.rollback();
        throw t;
    }
}

// ...
void setTransactionManager( TransactionManager tm ) { this.tm = tm; }
void setDataSourceFactory( DataSourceFactory dsf ) {
    left = dsf.createXADataSource( getLeftProperties() );
    right = dsf.createXADataSource( getRightProperties() );
}

In the previous example, the Transaction Manager service could have been injected with a component model like Declarative Services:

<reference interface="javax.transaction.TransactionManager" bind="setTransactionManager"/>
<reference name="dsf" interface="org.osgi.service.jdbc.DataSourceFactory" bind="setDataSourceFactory"/>

For example, it is possible to provide a Data Source service that provides automatic enlistment of the Connection as an XA Resource when one of its getConnection methods is called inside a transaction. The following code contains a Declarative Service component that implement this design. The component references a Transaction Manager service and a Data Source Factory service and provides a Data Source service that proxies an XA Data Source. Applications depend on the Data Source service, assuming that the Data Source service automatically enlists the connections it uses inside a transaction. See for an overview Figure 123.2 on page 384.

![Figure 123.2 Data Source Proxy](image_url)

This general purpose Data Source Proxy component can be fully configured by the Configuration Admin service to instantiate this component for each needed database connection. The Declarative Services service properties can be used to select a Data Source Factory for the required database driver (using the target), as well as provide the configuration properties for the creation of an XA Data Source. That is, such a component could be part of a support library.

The code for such an Application component could start like:
public class DataSourceProxy implements DataSource{
    Properties         properties  = new Properties();
    TransactionManager tm;
    XADataSource       xads;

    The activate method is called when the component's dependencies are met, that is, there is a Transaction Manager service as well as a matching Data Source Factory service. In this method, the properties of the component are copied to a Properties object to be compatible with the Data Source Factory factory methods.

    void activate(ComponentContext c) {
        // copy the properties set by the Config Admin into properties
        ...
    }

    The relevant methods in the Data Source Proxy component are the getConnection methods. The contract for this proxy component is that it enlists the XA Data Connection's XA Resource when it is called inside a transaction. This enlistment is done in the private enlist method.

    public Connection getConnection() throws SQLException{
        XAConnection  connection = xads.getXAConnection();
        return enlist(connection); }

    public Connection getConnection(String username, String password)
        throws SQLException {
        XAConnection connection = xads.getXAConnection(username,password);
        return enlist(connection); }

    The enlist method checks if there currently is a transaction active. If not, it ignores the enlistment, the connection will then not be connection to the transaction. If there is a current transaction, it enlists the corresponding XA Resource.

    private Connection enlist(XAConnection connection) throws SQLException {
        try {
            Transaction transaction = tm.getTransaction();
            if (transaction != null)
                transaction.enlistResource( connection.getXAResource());
        } catch (Exception e) {
            SQLException sqle=
                new SQLException("Failed to enlist");
            sqle.initCause(e);
            throw sqle;
        }
        return connection.getConnection();
    }

    What remains are a number of boilerplate methods that forward to the XA Data Source or set the dependencies.

    void setTransactionManager(TransactionManager tm) { this.tm = tm; }
    void setDataSourceFactory(DataSourceFactory dsf) throws Exception{
        xads = dsf.createXADataSource(properties);}
    public PrintWriter getLogWriter()
        throws SQLException { return xads.getLogWriter(); }

    public int getLoginTimeout()
throws SQLException { return xads.getLoginTimeout();}

public void setLogWriter(PrintWriter out)
    throws SQLException { xads.setLogWriter(out); }

public void setLoginTimeout(int seconds)
    throws SQLException { xads.setLoginTimeout(seconds);}

This is a fully coded example, it only lacks the configuration definitions for the Configuration Admin service.

This example Data Source proxy component makes it possible for an Application to depend on a Data Source service. The connections the Application uses from this Data Source are automatically transactional as long as there is a current transaction when the service is called. However, this approach only works when all bundles in the OSGi framework follow the same enlistment strategy because this specification does not provide a common enlistment strategy.

### 123.3 Container Managed Enlistment

The Application Container is responsible for enlisting Resource Managers used by the Application. For example, the Java EE Web and EJB Containers have a well defined model for managing resources within a transaction. If an Application runs inside a Java EE Container then it is the responsibility of the Java EE Container to handle the resource enlistment, the actual rules are beyond this specification.

A Transaction Originator Application bundle running inside a Container which manages any Resource Managers enlistment may use the User Transaction service for transaction demarcation, assuming this service is made available by the Container.

When a Java EE Container runs inside an OSGi framework then it must ensure that any services seen by its contained Applications are the same Transaction services as other bundles on that OSGi framework.

### 123.4 Resource Managers

Resource Managers perform work that needs to be committed or rolled back in a transaction. To participate in a transaction, a Resource Manager must have an XA Resource enlisted with the current transaction. This specification does not define how OSGi service implementations should be enlisted. This can be done by a Java EE Container, the Applications themselves, or through some other unspecified means.

### 123.5 The JTA Provider

The JTA Provider is the entity that provides the transaction services:

- **User Transaction** - A service that implements the JTA UserTransaction interface.
- **Transaction Manager** - A service that implements the JTA TransactionManager interface.
- **Transaction Synchronization Registry** - A service that implements the JTA TransactionSynchronizationRegistry interface.

There can be at most one JTA Provider in an OSGi framework and this JTA Provider must ensure that at most one transaction is associated with an application thread at any moment in time. All JTA Provider's transaction services must map to the same underlying JTA implementation. All JTA services should only be registered once.
123.5.1 User Transaction

The User Transaction service may be used by an Application bundle, acting as the Transaction Originator, to demarcate transaction boundaries when the bundle has no need to perform resource enlistment.

123.5.2 Transaction Manager

The Transaction Manager service offers transaction demarcation and further transaction management capabilities, such as Durable and Volatile resource enlistment, to an Application bundle or Application Container.

123.5.3 Transaction Synchronization Service

The Transaction Synchronization Registry service may be used by an Application bundle or a Container. The service provides for the registration of Volatile Resources that implement the JTA Synchronization interface.

For example:

```java
private class MyVolatile implements Synchronization{
    // ...
}
TransactionSynchronizationRegistry tsr = ...; // may be injected
tsr.registerInterposedSynchronization(new MyVolatile());
```

123.6 Life Cycle

123.6.1 JTA Provider

The life cycle of the transaction services and bundles that make up the JTA Provider must be dealt with appropriately such that implementations always ensure the atomic nature of transactions. When the JTA Provider is stopped and its services are unregistered, the JTA Provider must make sure that all active transactions are dealt with appropriately. A JTA Provider can decide to rollback all active transactions or it can decide to keep track of existing active transactions and allow them to continue to their normal conclusion but not allow any new transactions to be created. Any failures caused by executing code outside their life cycle can be dealt with as general failures. From a transactional consistency point of view, stopping the bundle(s) that implement the JTA Provider while transactional work is in-flight, is no different from a failure of the framework hosting the JTA Provider. In either case transaction recovery is initiated by the JTA Provider after it has re-started.

There are well-defined XA semantics between a JTA Provider and Resource Managers in the event of a failure of either at any point in a transaction. If a Resource Manager bundle is stopped while it is involved in-flight transactions then the JTA Provider should exhibit the same external behavior it does in the event of a communication failure with the Resource Manager. For example a JTA Provider will respond to an XAER_RMFAIL response resulting from calling the XAResource commit method by retrying the commit. The mechanism used by the JTA Provider to determine when to retry the commit is a detail of the implementation.

123.6.2 Application Bundles

Applications can act in the role of the Transaction Originator. There is no guarantee that an Application that starts a transaction will always be available to complete the transaction since the client can fail independently of the JTA Provider. A failure of the Application Bundle to complete, in a timely fashion, a transaction it originated must finally result in the JTA Provider rolling back the transaction.
123.6.3 Error Handling

This specification does not define a specific error handling strategy. Exceptions and errors that occur during transaction processing can result in the transaction being marked *rollback-only* by the container or framework in which an Application runs or may be left for the Application to handle. An Application which receives an error or an exception while running under a transaction can choose to mark the transaction rollback-only.

123.7 Security

This specification relies on the security model of JTA.

123.8 References

[1] *Java Transaction API Specification 1.1*  

[2] *XA+ Specification*  

[3] *Transaction Processing*  

http://jcp.org/en/jsr/detail?id=244
### 124 JMX™ Management Model Specification

#### 124.1 Introduction

The Java Management Extensions (JMX) is the standard API specification for providing a management interface to Java SE and Java EE applications. The JMX specification defines the design patterns, APIs, services and architecture for application management, network management and monitoring in the Java programming language. The need to administer, monitor and manage a container is today recognized as a prerequisite in the enterprise software domain.

While OSGi defines a rich API for controlling all aspects of the framework, this API is not suitable for direct usage in the JMX framework because it was not designed to be remoted. This specification provides an interface adaptation of the existing OSGi framework, which can be used to expose an OSGi Framework manipulation API to any JMX compliant implementation. Interfaces and system semantics for a monitoring system are specified for exposing the underlying artifacts of the OSGi framework such as services and bundles. Additionally, the management of a number of core and compendium services have been standardized in this document.

Finally, a standardized JMX object naming standard is proposed so that management objects are uniformly named across implementations such that any JMX compliant system can find, manipulate and interact with the framework and artifacts that it manages.

This specification requires version 1.2 or later of JMX, which implies the use of Java 5.

#### 124.1.1 Essentials

- **Life Cycle** - Must allow support of full life cycle management of bundles.
- **Batch** - Support batch oriented operations to minimize the influence of network capacity and latency.
- **Compatible** - This specification must work naturally with JMX.
- **Efficient** - Minimize the number of registered objects to not overload the MBean Server and communication channels.
- **Open MBean** - Support the Open MBean layer of JMX instead of using domain specific objects.
- **Core** - Supports all the Framework's operations.
- **Core Services** - Support the framework services if registered, except for Conditional Permission Admin.

#### 124.1.2 Entities

- **MBean** - A Managed Bean. The core concept of JMX to manage an entity.
- **MBean Server** - The MBean Server is the access point for registering MBeans.
- **Manager** - The entity that implements the MBeans and registers them with the registered MBean servers.
- **ObjectName** - A name for an MBean registered with an MBean Server.
- **Bundle State MBean** - Provides central access to the state of a bundle in a framework.
- **Framework MBean** - Represents the general framework's state and can be used to manage the life cycle of bundles.
- **Bundle Wiring State MBean** - Provides access to the wiring state of the framework.
- **Service State MBean** - Provides access to the service information in the service registry. It provides both a general MBean interface as well as an Open Type description.
- **Configuration Admin MBean** - Can be used to manipulate a Configuration Admin service.
- **Permission Admin MBean** - Provides access to the Permission Admin service.
- **Provisioning Service MBean** - Provides access to the Provisioning Service.
- **User Admin MBean** - Provides access to the User Admin service.
- **Item** - A helper class to create Open Types. This class is intended to make the Javadoc easier to navigate and keep definitions close together. This is otherwise hard to do with Open Type. This class has no utility for management applications.
- **Open Type** - A JMX metadata standard to describe MBeans.
- **Remote Manager** - The entity accessing a MBean Server remotely.
- **JConsole** - The default Java Remote Manager.

**Figure 124.1** MBeans

**124.1.3 Synopsis**

This specification plays a part in both the OSGi framework as well as in a remote manager.

A JMX OSGi manager bundle obtains one or more MBean servers that are registered as services. The JMX OSGi manager then registers all its managed beans: Framework MBean, Bundle State MBean, Package State MBean, and the Service State MBean under their JMX object names. If a number of optional services are registered, then the JMX OSGi bundle must also register a corresponding MBean with the MBean server for each of the services that it can obtain.

A remote manager can access an MBean Server running in a (remote) VM. The remote manager can then discover any MBeans. These MBeans can be manipulated as dynamic types or as specific types as outlined in this specification.
124.2 **JMX Overview**

JMX is a specification which defines how arbitrary remote communication protocols and mechanisms can be adapted to interact with the underlying management APIs exposed by JMX compliant implementations. JMX is not a remote communication standard, the actual protocols can vary. The JMX architecture is composed of three levels:

- **Instrumentation** - The managed resources of the system are instrumented using managed beans (a.k.a. MBeans) which expose their management interfaces through a JMX agent for remote management and monitoring.

- **Agent** - The JMX agent layer is mainly represented by the MBean server. This is the managed object server where the MBeans are registered. The JMX agent includes a set of functions for manipulating the registered MBeans, which directly expose and control the underlying resources, and then make them available to remote managers.

- **Remote Manager** - The remote management layer provides the specification for the actual remote communication protocol adapters and defines standard connectors which make the JMX agent accessible to remote managers outside of the Java process that hosts the agent.

The JMX Architecture is depicted in Figure 124.2.

![JMX Architecture Diagram](image)

124.2.1 **Connectors and Adapters**

Connectors are used to connect an agent with a remote JMX-enabled managers. This form of communication involves a connector in the JMX agent and a connector client in the management application. Protocol adapters provide a management view of the JMX agent through a given protocol.

Remote managers that connect to a protocol adapter are usually specific to the given protocol. Remote Managers can be generic consoles (such as JConsole; see [8] Using JConsole to Monitor Applications), or domain-specific monitoring applications. External applications can interact with the MBeans through the use of JMX connectors and protocol adapters.

124.2.2 **Object Name**

All managed objects in JMX are referenced via JMX Object Names. Object Names are strings which can be resolved within the context of a JMX MBean Server in order. An Object Name consists of two parts:

```
ObjectName ::= domain ':' properties
properties ::= property ( ',' property )*  
```

To avoid collisions between MBeans supplied by different vendors, a recommended convention is to begin the domain name with the reverse DNS name of the organization that specifies the MBeans, followed by a full stop (‘.’) and a string whose interpretation is determined by that organization.
MBeans specified by the OSGi Alliance have domains that start with osgi.

### 124.2.3 MBeans

Any object can be registered with an MBean Server and manipulated remotely over an *MBean Server Connection*. An MBean Server Connection can represent a local MBean Server or a remote MBean Server. An MBean is always identified by an *Object Name*. The Object Name identifies a remote MBean uniquely within a specific MBean Server Connection.

Standard manipulations of a remote MBean are done through *attributes* and *operations*, which are similar to properties and methods for Java beans. Not all methods on the implementation class can be used, the registering party must specifically provide access to the methods that can be called remotely. The registrar can define the exposed operations with the following mechanisms:

- **Design Pattern** - Let the registered object implement an *MBean interface* that has the fully qualified name of the implementation class suffixed with *MBean*. The MBean server will then limit access to attributes and properties defined in the MBean interface. For example, the `com.acme.Resource` class should implement the `com.acme.ResourceMBean` interface. The `com.acme.ResourceMBean` interface would define the properties and operations.

- **Dynamic MBean** - Register a Dynamic MBean, which handles the access to the operations and attributes programatically. The JMX specification provides the `DynamicMBean` interface for this purpose. If the MBean registered with an MBean Server implements this interface, then the MBean Server must get the MBean's metadata through the `DynamicMBean` interface instead of using reflection. Therefore, Dynamic MBeans can provide more rich metadata that describes their operations and attributes.

- **Standard MBean** - Register a Standard MBean. A standard MBean works the same as the previous bullet but does not require the implementation class name to map to the MBean interface name.

Attributes map to properties on the registered MBean interface and operations allow the invocation of an arbitrary method on the remote MBean with arbitrary parameters. The following code example shows how to get the size property of a remote MBean in this way:

```java
void drop(MBeanServerConnection mbs, ObjectName objectName) {
    Integer sizeI = (Integer)
            mbs.getAttribute(objectName, "Size");
    int size = sizeI.intValue();
    if (size > desiredSize) {
        mbs.invoke(objectName, "dropOldest",
            new Integer[] {new Integer(size - desiredSize)},
            new String[] {"int"});
    }
}
```

In release 1.2 the JMX specification introduced the *MBean Server Invocation Handler* to simplify the manipulation of the remote MBeans by creating a *proxy* for an *MBean interface* that implements all the relevant methods. An MBean interface defines the methods and properties for an MBean. The proxy has a reference to an *MBean Server Connection*, it can therefore automate the invocation of the appropriate methods from the MBean interface. Therefore, by using an MBean interface, it is possible to simplify the remote manager:

```java
MBeanServer mbs = ...;
CacheControlMBean cacheControl = (CacheControlMBean)
        MbeanServerInvocationHandler.newProxyInstance(
            mbs, objectName, CacheControlMBean.class, false);

int size = cacheControl.getSize();
if (size > desiredSize) {
    // ...
}
```
cacheControl.dropOldest(size - desiredSize);

The creation of the proxy is somewhat verbose, but once it is available, the MBean can be accessed like a local object. The proxy is much easier to use and read, and much less error-prone, than accessing the MBean Server method through invoking operations and getting attributes.

The MBean interface can also ensure a certain amount of type safety. The MBean implementation can implement the MBean interface and the remote manager uses the proxy implementing this interface. However, neither is required. The MBean can directly implement the methods without implementing the interface and the remote manager can directly manipulate the attributes and invocations.

The key advantage is therefore the documentation of the management interface. Using an MBean interface, this can be done very concisely and it allows the usage of standard tools for Java source code and Javadoc.

124.2.4 Open Types

The distributed nature of remote management poses a number of problems for exchanging general objects.

- **Versioning** - All participating parties require access to the same version of the object's class.
- **Serialization** - Not all objects are easy to serialize.
- **Size** - Arbitrary objects can transitively link to large amounts of data.
- **Descriptive** - Classes provide little or no support for editing.
- **Limited** - Classes are Java specific, making it harder to interact with non-Java environments.

An alternative is to limit the management types to be exchanged to small, well defined set. Open MBeans limit the used data types to small number of types called the *basic types*. These types are supported by all JMX 1.2 and later implementations. This basic set of types contains:

- **Primitives** - boolean, byte, char, short, int, long, float, double.
- **Primitive Arrays** - boolean[], byte[], char[], short[], int[], long[], float[], double[].
- **Wrappers** - Boolean, Byte, Character, Short, Integer, Long, Float, Double.
- **Scalars** - String, BigDecimal, BigInteger, Date, ObjectName.
- **Complex** - CompositeData, TabularData, and complex arrays.
- **Return** - Void, operation return only.

The Complex types are unique to JMX, they are used to provide access to complex data (like objects) without using classes. The complex types are *self describing*. The metadata associated with these complex types allow a remote manager to discover the structure and automatically construct a (graphic) user interface for these complex objects.

Open MBeans must be Dynamic MBeans when registered. Furthermore, they must provide Open MBean variations of the Info objects that describe the operations and attributes.

124.3 OSGi JMX Management

The OSGi JMX Management model is based on Open MBeans, see Open Types on page 393. This specification declares a number of MBeans for the core Framework, some of the core services, and a number of compendium services. Though Open MBeans are based on Dynamic MBeans, this specification uses the traditional MBean interface to define the management interaction patterns. The implementer of this specification must register an implementation of these interfaces as a Dynamic MBean. An implementation should provide the additional Open MBeans Info objects for the operations and attributes.
This specification defines the following Open MBeans:

- **Core Framework** - FrameworkMBean, BundleStateMBean, ServiceStateMBean, BundleWiringStateMBean, and PackageStateMBean.
- **Core Services** - PermissionAdminMBean. The Conditional Permission Admin is not included in this specification.
- **Compendium Services** - ConfigurationAdminMBean, UserAdminMBean, ProvisioningServiceMBean

### 124.3.1 Naming

The MBean interfaces have been named after the service they manage. That is the ConfigurationAdminMBean interface manages the Configuration Admin service, which is modelled with the ConfigurationAdmin interface.

Package names are constructed from taking the corresponding resource package and inserting jmx. after org.osgi. For example

```text
org.osgi.framework          org.osgi.jmx.framework
org.osgi.service.cm         org.osgi.jmx.service.cm
```

It is not possible to use the MBean interface design pattern because the MBean interfaces are in OSGi packages. The design pattern requires the fully qualified name of the implementation suffixed with MBean to match the MBean interface name. This would require that the implementation class resides in an OSGi package, which would extend these packages.

However, the StandardMBean class allows the association of one of the OSGi MBean interfaces with an arbitrary class.

### 124.3.2 Object Naming

Object Names for OSGi managed MBeans must follow the following structure:

```text
object-name := ( core | compendium )
            ,version=' version
            ,framework=' framework
            ,uuid=' uuid
            ( , ' key '=' value )*
core := 'osgi.core:' framework-type
compendium := 'osgi.compendium:' service-type
framework-type := ( 'type=' token ) | service-type
service-type := 'service=' token
framework := <Bundle symbolic name of the system bundle>
uuid := <org.osgi.framework.uuid Framework property's value>
key := <any jmx supported key>
value := <any jmx supported value>
```

There are the following additional constraints:

- **Spaces** - Spaces between any of the terminals are not permitted.
- **Version** - The version must be limited to a major and minor version part. The given version must identify the package of the corresponding resource. For example, if the Configuration Admin service is on version 1.3.2.200910101250, then the version in the Object Name must be 1.3.
- **Service** - The service-type should use the package name of the corresponding service. For example, for Configuration Admin this would be service=cm.

The Object Name must contain the framework bundle symbolic name and its UUID so that multiple instances on the same VM can be discriminated. An example of an Object Name is:

```text
osgi.core:type=framework,version=1.7,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. Patterns are names have an asterisk ('*' or '*'). For instance, the following query allows a client to find all Framework MBeans for an Apache Felix implementation without having to rely on knowing the UUID:

```java
ObjectName on = new ObjectName("osgi.core:type=framework," + "version=1.7,framework=org.apache.felix.framework,*");
Set<ObjectInstance> instances = mserver.queryMBeans(on,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:

```java
ObjectName on = new ObjectName("osgi.core:type=framework,version=1.7,*");
Set<ObjectInstance> instances = mserver.queryMBeans(on,null);
```

The `uuid` and `framework` property keys are only applicable to OSGi JMX Management Model Specification Version 1.1 and above.

To maintain backward compatibility, a OSGi JMX Framework package Version 1.7 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.

The actual object name prefixes are defined in the MBean interfaces. For example, the Object Name for the Configuration Admin MBean is:

```
osgi.compendium:service=cm,version=1.3
```

It is the responsibility of the party registering the MBean to suffix this with the framework and UUID.

In this specification, all management interfaces are specified to return opaque Strings or longs rather than Object Names so that the MBean interfaces contain no JMX specific artifacts and can be used with a variety of remote access protocols such as SNMP, etc. Non JMX use of these APIs can use these Strings as their own opaque identifiers without any change to the interfaces themselves.

### 124.3.3 The MBean Server

An implementation of this specification must find all MBean Servers services that it has access to. It should then register all MBeans with each server found in the service registry.

A compliant implementation must register all the framework’s MBeans: FrameworkMBean, BundleStateMBean, ServiceStateMBean, BundleWiringStateMBean and PackageStateMBean. The registration of the compendium services is optional. However, if they are registered they must implement the behavior as defined in this specification.

### 124.3.4 Registrations

The OSGi MBeans are designed to minimize the notifications. That is, the objects model a command interface to access the required information. Their registration is not intended to signify anything else than the start of the manager bundle and the availability of the underlying resource.

Implementations must always register only one of each of the Framework MBean types (Framework MBean, Service State MBean, Bundle State MBean, Wiring State MBean, and Package State MBean). All other MBean types depend on the registered services they manage. Each service requires its unique MBean. If no corresponding service is present, then no MBean should be registered. Modified events must be ignored. If a manager supports a specific OSGi MBean for a compendium service then it must register an MBean for each instance of that service.
124.4 MBeans

This specification defines MBean interfaces listed in the following table. The Object Name specified in this table is broken into a number of lines for readability, however, newlines and whitespace is not allowed in the Object Name.

<table>
<thead>
<tr>
<th>MBean</th>
<th>Object Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| FrameworkMBean         | osgi.core:
                           type=framework,
                           version=1.7                                                               | Provides access to bundle life cycle methods of the framework including batch install and update operations. |
| BundleStateMBean       | osgi.core:
                           type=.bundleState,                                                      | Provides detailed access to the state of one bundle and aggregated state of a group of bundles. |
| ServiceStateMBean      | osgi.core:
                           type=serviceState,                                                      | Provides detailed access to the state of one service and aggregated state of a group of services. |
| PackageStateMBean      | osgi.core:
                           type=packageState,                                                      | Provides detailed access to the state of one package and aggregated state of a group of packages. |
| PermissionAdminMBean   | osgi.core:
                           service=permissionadmin,                                                | Based on the Permission Admin service.                                            |
| ConfigurationAdminMBean| osgi.compendium:
                           service=cm,                                                               | Manages a Configuration Admin service.                                           |
| ProvisioningServiceMBean| osgi.compendium:
                           service=provisioning,                                                   | Manages a Provisioning Service.                                                  |
| UserAdminMBean         | osgi.compendium:
                           service=useradmin,                                                      | Manages a User Admin service.                                                    |
| BundleWiringStateMBean | osgi.core:
                           service=wiringState,                                                    | Reflects the Framework’s wiring state.                                           |

124.5 Item

The MBean interfaces do not only define the Java interface, they also define the Open Types. These types are defined with the Item class in this specification to simplify the definitions; the Item class has no role in a management application. The Item class is used to allow the items used in Compos-
ite Types to be encoded in the interface. This is not possible with the standard Open Types because they use exceptions and use parallel arrays. For example, the following code defines a static Open Type without the Item class:

```java
static CompositeType HEADER;
static {
    try {
        HEADER = new CompositeType("HEADER" "This is a header",
                               new String[] {"KEY", "VALUE"},
                               new String[] {"A key for a header", "A value for a header"},
                               new OpenType[] { SimpleType.STRING, SimpleType.STRING });
        catch(OpenDataException e) {
            ...
        }
    }
}
```

This code can be replaced with the Item class:

```java
static Item KEY = new Item("KEY", "A key for header", SimpleType.STRING );
static Item VALUE = new Item("VALUE", "A value for header", SimpleType.STRING );
static CompositeType HEADER = Item.composite("HEADER", "This is a header",
                                           KEY, VALUE );
```

The Item class also provides a number of convenience methods to construct the different Open Types. However, the intention is to simplify the specification definitions, not as an aid in management operations.

### 124.6 Security

Exposing any system remotely opens up a, potentially, devastating security hole in a system. Remote entities should establish their identity and the management system should be able to control the access these entities have over the management system. JMX seamlessly inter operates with the Java Authentication and Authorization Service (JAAS) and Java 2 platform Standard Edition (Java SE) Security Architecture.

The JMX OSGi manager must have access to the services it manages and the operations it invokes. It is likely that this bundle requires All Permission because it needs to invoke operations on the Conditional Permission Admin. It is strongly advised that implementations limit the set of available permissions based on authenticating the remote manager.

### 124.7 org.osgi.jmx

OSGi JMX Package Version 1.1.

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:

```ini
Import-Package: org.osgi.jmx; version="[1.1,2.0)"
```

Example import for providers implementing the API in this package:

```ini
Import-Package: org.osgi.jmx; version="[1.1,1.2)"
```
124.7.1 Summary

- Item - The item class enables the definition of open types in the appropriate interfaces.
- JmxConstants - Constants for OSGi JMX Specification.

124.7.2 public class Item

The item class enables the definition of open types in the appropriate interfaces. This class contains a number of methods that make it possible to create open types for CompositeType, TabularType, and ArrayType. The normal creation throws a checked exception, making it impossible to use them in a static initializer. The constructors are also not very suitable for static construction. An Item instance describes an item in a Composite Type. It groups the triplet of name, description, and Open Type. These Item instances allows the definitions of an item to stay together.

Concurrency Immutable

124.7.2.1 public Item(String name,String description,OpenType type,String ... restrictions)

name The name of the item.
description The description of the item.
type The Open Type of this item.
restrictions Ignored, contains list of restrictions

□ Create a triple of name, description, and type. This triplet is used in the creation of a Composite Type.

124.7.2.2 public static ArrayType arrayType(int dim,OpenType elementType)

dim The dimension
elementType The element type

□ Return a new Array Type.

Returns A new Array Type

124.7.2.3 public static CompositeType compositeType(String name,String description,Item ... items)

name The name of the Tabular Type.
description The description of the Tabular Type.
items The items that describe the composite type.

□ Create a Composite Type

Returns a new Composite Type

Throws RuntimeException – when the Tabular Type throws an OpenDataException

124.7.2.4 public static CompositeType extend(CompositeType parent,String name,String description,Item ... items)

parent The parent type, can be null
name The name of the type
description The description of the type
items The items that should be added/override to the parent type

□ Extend a Composite Type by adding new items. Items can override items in the parent type.

Returns A new Composite Type that extends the parent type

Throws RuntimeException – when an OpenDataException is thrown
124.7.2.5  public static TabularType tabularType(String name, String description, CompositeType rowType, String ... index)

  name  The name of the Tabular Type.
  description  The description of the Tabular Type.
  rowType  The Open Type for a row
  index  The names of the items that form the index.

Create a Tabular Type.

Returns  A new Tabular Type composed from the parameters.

Throws  RuntimeException – when the Tabular Type throws an OpenDataException

124.7.3  public class JmxConstants

Constants for OSGi JMX Specification. Additionally, this class contains a number of utility types that are used in different places in the specification. These are LONG_ARRAY_TYPE, STRING_ARRAY_TYPE, and PROPERTIES_TYPE.

Concurrency  Immutable

124.7.3.1  public static final String ARRAY_OF = "Array of ">

For an encoded array we need to start with ARRAY_OF. This must be followed by one of the names in SCALAR.

124.7.3.2  public static final String BIGDECIMAL = "BigDecimal"

Value for PROPERTY_TYPE value in the case of java.math.BigDecimal

124.7.3.3  public static final String BIGINTEGER = "BigInteger"

Value for PROPERTY_TYPE value in the case of java.math.BigInteger

124.7.3.4  public static final String BOOLEAN = "Boolean"

Value for PROPERTY_TYPE value in the case of java.lang.Boolean

124.7.3.5  public static final String BYTE = "Byte"

Value for PROPERTY_TYPE value in the case of java.lang.Byte

124.7.3.6  public static final String CHARACTER = "Character"

Value for PROPERTY_TYPE value in the case of java.lang.Character

124.7.3.7  public static final String DOUBLE = "Double"

Value for PROPERTY_TYPE value in the case of java.lang.Double

124.7.3.8  public static final String FLOAT = "Float"

Value for PROPERTY_TYPE value in the case of java.lang.Float

124.7.3.9  public static final String INTEGER = "Integer"

Value for PROPERTY_TYPE value in the case of java.lang.Integer

124.7.3.10  public static final String KEY = "Key"

The key KEY.

124.7.3.11  public static final Item KEY_ITEM

The key of a property. The key is KEY and the type is SimpleType.STRING.
124.7.3.12 public static final String LONG = "Long"
Value for PROPERTY_TYPE value in the case of java.lang.Long

124.7.3.13 public static final ArrayType LONG_ARRAY_TYPE
The MBean Open type for an array of longs

124.7.3.14 public static final String OSGI_COMPENDIUM = "osgi.compendium"
The domain name of the selected OSGi compendium MBeans

124.7.3.15 public static final String OSGI_CORE = "osgi.core"
The domain name of the core OSGi MBeans

124.7.3.16 public static final String P_BOOLEAN = "boolean"
Value for PROPERTY_TYPE value in the case of the boolean primitive type.

124.7.3.17 public static final String P_BYTE = "byte"
Value for PROPERTY_TYPE value in the case of the byte primitive type.

124.7.3.18 public static final String P_CHAR = "char"
Value for PROPERTY_TYPE value in the case of the char primitive type.

124.7.3.19 public static final String P_DOUBLE = "double"
Value for PROPERTY_TYPE value in the case of the double primitive type.

124.7.3.20 public static final String P_FLOAT = "float"
Value for PROPERTY_TYPE value in the case of the float primitive type.

124.7.3.21 public static final String P_INT = "int"
Value for PROPERTY_TYPE value in the case of the int primitive type.

124.7.3.22 public static final String P_LONG = "long"
Value for PROPERTY_TYPE value in the case of the long primitive type.

124.7.3.23 public static final String P_SHORT = "short"
Value for PROPERTY_TYPE value in the case of the short primitive type.

124.7.3.24 public static final TabularType PROPERTIES_TYPE
Describes a map with properties. The row type is PROPERTY_TYPE. The index is defined to the KEY of the property.

124.7.3.25 public static final CompositeType PROPERTY_TYPE
A Composite Type describing a single property. A property consists of the following items KEY_ITEM, VALUE_ITEM, and TYPE_ITEM.

124.7.3.26 public static final List<String> SCALAR
A set of all scalars that can be used in the TYPE property of a PROPERTIES_TYPE. This contains the following names:

- BIGDECIMAL
- BIGINTEGER
- BOOLEAN
- BYTE
- CHARACTER
- DOUBLE
- FLOAT
- INTEGER
- LONG
- SHORT
- STRING
- VERSION
- P_BYTE
- P_CHAR
- P_DOUBLE
- P_FLOAT
- P_INT
- P_LONG
- P_SHORT

124.7.3.27  
public static final String SHORT = "Short"
Value for PROPERTY_TYPE value in the case of java.lang.Short

124.7.3.28  
public static final String STRING = "String"
Value for PROPERTY_TYPE value in the case of java.lang.String

124.7.3.29  
public static final ArrayType STRING_ARRAY_TYPE
The MBean Open type for an array of strings

124.7.3.30  
public static final String TYPE = "Type"
The key TYPE.

124.7.3.31  
public static final Item TYPE_ITEM
The type of the property. The key is TYPE and the type is SimpleType.STRING. This string must follow the following syntax:

```
type ::= scalar | vector | array
vector ::= 'Vector of' scalar
array ::= 'Array of' (scalar | primitive)
scalar ::= 'String' | 'BigInteger' | 'BigDecimal'
        | 'Byte' | 'Character' | 'Short'
        | 'Integer' | 'Long' | 'Float'
        | 'Double' | 'Version'
primitive ::= 'byte' | 'char' | 'short'
           | 'int' | 'long' | 'float'
           | 'double'
```

This encoding does not support arrays in vectors or arrays. Arrays and vectors can only contain scalars. List properties are encoded as arrays. Empty lists, arrays or vectors are not represented. Null is not an allowed value.
For example, the encoding of a byte array `byte[] {1,2,3,5,7}` would look like:

```plaintext
type: 'Array of byte'
value: 1,2,3,5,7
```

Quoting can be used as follows:

```plaintext
type: 'Array of String'
value: 'abc', 'def', '\'quoted\'', '"quoted"', '\'
```

```java
public static final String VALUE = "Value"
```

The key VALUE.

```java
public static final Item VALUE_ITEM
```

The value of a property. The key is VALUE and the type is SimpleType.STRING. A value will be encoded by the string given in TYPE. The syntax for this type is given in TYPE_ITEM.

```java
public static final String VECTOR_OF = "Vector of"
```

For an encoded vector we need to start with ARRAY_OF. This must be followed by one of the names in SCALAR.

```java
public static final String VERSION = "Version"
```

Value for PROPERTY_TYPE value in the case of Version. Since 1.1

### 124.8 org.osgi.jmx.framework

OSGi JMX Framework Package Version 1.7.

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:

```text
Import-Package: org.osgi.jmx.framework; version="[1.7,2.0)"
```

Example import for providers implementing the API in this package:

```text
Import-Package: org.osgi.jmx.framework; version="[1.7,1.8)"
```

#### 124.8.1 Summary

- BundleStateMBean - This MBean represents the Bundle state of the framework.
- FrameworkMBean - The FrameworkMBean provides mechanisms to exert control over the framework.
- PackageStateMBean - This MBean provides information about the package state of the framework.
- ServiceStateMBean - This MBean represents the Service state of the framework.

#### 124.8.2 public interface BundleStateMBean

This MBean represents the Bundle state of the framework. This MBean also emits events that clients can use to get notified of the changes in the bundle state of the framework.
Concurrency: Thread-safe

124.8.2.1  
```java
public static final String ACTIVATION_POLICY_USED = "ActivationPolicyUsed"
```

The key ACTIVATION_POLICY_USED, used in ACTIVATION_POLICY_USED_ITEM.

124.8.2.2  
```java
public static final Item ACTIVATION_POLICY_USED_ITEM
```

The item containing the indication whether the bundle activation policy must be used in BUNDLE_TYPE. The key is ACTIVATION_POLICY_USED and the type is SimpleType.BOOLEAN.

124.8.2.3  
```java
public static final String ACTIVE = "ACTIVE"
```

Constant ACTIVE for the STATE.

124.8.2.4  
```java
public static final CompositeType BUNDLE_EVENT_TYPE
```

The Composite Type that represents a bundle event. This composite consists of:

- IDENTIFIER
- LOCATION
- SYMBOLIC_NAME
- EVENT

124.8.2.5  
```java
public static final CompositeType BUNDLE_TYPE
```

The Composite Type that represents a bundle. This composite consist of:

- EXPORTED_PACKAGES
- FRAGMENT
- FRAGMENTS
- HEADERS
- HOSTS
- IDENTIFIER
- IMPORTED_PACKAGES
- LAST_MODIFIED
- LOCATION
- ACTIVATION_POLICY_USED
- PERSISTENTLY_STARTED
- REGISTERED_SERVICES
- REMOVAL_PENDING
- REQUIRED
- REQUIRED_BUNDLES
- REQUIRING_BUNDLES
- START_LEVEL
- STATE
- SERVICES_IN_USE
- SYMBOLIC_NAME
- VERSION

It is used by BUNDLES_TYPE.

124.8.2.6  
```java
public static final TabularType BUNDLES_TYPE
```

The Tabular Type for a list of bundles. The row type is BUNDLE_TYPE and the index is IDENTIFIER.
124.8.2.7 public static final String EVENT = "BundleEvent"
   The key EVENT, used in EVENT_ITEM.

124.8.2.8 public static final Item EVENT_ITEM
   The item containing the event type. The key is EVENT and the type is SimpleType.INTEGER

124.8.2.9 public static final String EXPORTED_PACKAGES = "ExportedPackages"
   The key EXPORTED_PACKAGES, used in EXPORTED_PACKAGES_ITEM.

124.8.2.10 public static final Item EXPORTED_PACKAGES_ITEM
   The item containing the exported package names in BUNDLE_TYPE. The key is EXPORTED_PACKAGES and the type is JmxConstants.STRING_ARRAY_TYPE.

124.8.2.11 public static final String FRAGMENT = "Fragment"
   The key FRAGMENT, used in FRAGMENT_ITEM.

124.8.2.12 public static final Item FRAGMENT_ITEM
   The item containing the fragment status in BUNDLE_TYPE. The key is FRAGMENT and the type is SimpleType.BOOLEAN.

124.8.2.13 public static final String FRAGMENTS = "Fragments"
   The key FRAGMENTS, used in FRAGMENTS_ITEM.

124.8.2.14 public static final Item FRAGMENTS_ITEM
   The item containing the list of fragments the bundle is host to in BUNDLE_TYPE. The key is FRAGMENTS and the type is JmxConstants.LONG_ARRAY_TYPE.

124.8.2.15 public static final CompositeType HEADER_TYPE
   The Composite Type describing an entry in bundle headers. It consists of KEY_ITEM and VALUE_ITEM.

124.8.2.16 public static final String HEADERS = "Headers"
   The key HEADERS, used in HEADERS_ITEM.

124.8.2.17 public static final Item HEADERS_ITEM
   The item containing the bundle headers in BUNDLE_TYPE. The key is HEADERS and the type is HEADERS_TYPE.

124.8.2.18 public static final TabularType HEADERS_TYPE
   The Tabular Type describing the type of the Tabular Data value that is returned from getHeaders(long) method. The primary item is KEY_ITEM.

124.8.2.19 public static final String HOSTS = "Hosts"
   The key HOSTS, used in HOSTS_ITEM.

124.8.2.20 public static final Item HOSTS_ITEM
   The item containing the bundle identifiers representing the hosts in BUNDLE_TYPE. The key is HOSTS and the type is JmxConstants.LONG_ARRAY_TYPE.

124.8.2.21 public static final String IDENTIFIER = "Identifier"
   The key IDENTIFIER, used in IDENTIFIER_ITEM.
124.8.2.22  public static final Item IDENTIFIER_ITEM
The item containing the bundle identifier in BUNDLE_TYPE. The key is IDENTIFIER and the type is SimpleType.LONG.

124.8.2.23  public static final String IMPORTED_PACKAGES = "ImportedPackages"
The key IMPORTED_PACKAGES, used in EXPORTED_PACKAGES_ITEM.

124.8.2.24  public static final Item IMPORTED_PACKAGES_ITEM
The item containing the imported package names in BUNDLE_TYPE. The key is IMPORTED_PACKAGES and the type is JmxConstants.STRING_ARRAY_TYPE.

124.8.2.25  public static final String INSTALLED = "INSTALLED"
Constant INSTALLED for the STATE.

124.8.2.26  public static final String KEY = "Key"
The key KEY, used in KEY_ITEM.

124.8.2.27  public static final Item KEY_ITEM
The item describing the key of a bundle header entry. The key is KEY and the type is SimpleType.STRING.

124.8.2.28  public static final String LAST_MODIFIED = "LastModified"
The key LAST_MODIFIED, used in LAST_MODIFIED_ITEM.

124.8.2.29  public static final Item LAST_MODIFIED_ITEM
The item containing the last modified time in the BUNDLE_TYPE. The key is LAST_MODIFIED and the type is SimpleType.LONG.

124.8.2.30  public static final String LOCATION = "Location"
The key LOCATION, used in LOCATION_ITEM.

124.8.2.31  public static final Item LOCATION_ITEM
The item containing the bundle location in BUNDLE_TYPE. The key is LOCATION and the type is SimpleType.STRING.

124.8.2.32  public static final String OBJECTNAME = "osgi.core:type=bundleState,version=1.7"
The Object Name prefix for this mbean. The full object name also contains the framework name and uuid as properties.

124.8.2.33  public static final String PERSISTENTLY_STARTED = "PersistentlyStarted"
The key PERSISTENTLY_STARTED, used in PERSISTENTLY_STARTED_ITEM.

124.8.2.34  public static final Item PERSISTENTLY_STARTED_ITEM
The item containing the indication of persistently started in BUNDLE_TYPE. The key is PERSISTENTLY_STARTED and the type is SimpleType.BOOLEAN.

124.8.2.35  public static final String REGISTERED_SERVICES = "RegisteredServices"
The key REGISTERED_SERVICES, used in REGISTERED_SERVICES_ITEM.

124.8.2.36  public static final Item REGISTERED_SERVICES_ITEM
The item containing the registered services of the bundle in BUNDLE_TYPE. The key is REGISTERED_SERVICES and the type is JmxConstants.LONG_ARRAY_TYPE.
public static final String REMOVAL_PENDING = "RemovalPending"
The key REMOVAL_PENDING, used in REMOVAL_PENDING_ITEM.

public static final Item REMOVAL_PENDING_ITEM
The item containing the indication of removal pending in BUNDLE_TYPE. The key is REMOVAL_PENDING and the type is SimpleType.BOOLEAN.

public static final String REQUIRED = "Required"
The key REQUIRED, used in REQUIRED_ITEM.

public static final String REQUIRED_BUNDLES = "RequiredBundles"
The key REQUIRED_BUNDLES, used in REQUIRED_BUNDLES_ITEM.

public static final Item REQUIRED_BUNDLES_ITEM
The item containing the required bundles in BUNDLE_TYPE. The key is REQUIRED_BUNDLES and the type is JmxConstants.LONG_ARRAY_TYPE

public static final Item REQUIRED_ITEM
The item containing the required status in BUNDLE_TYPE. The key is REQUIRED and the the type is SimpleType.BOOLEAN.

public static final String REQUIRING_BUNDLES = "RequiringBundles"
The key REQUIRING_BUNDLES, used in REQUIRING_BUNDLES_ITEM.

public static final Item REQUIRING_BUNDLES_ITEM
The item containing the bundles requiring this bundle in BUNDLE_TYPE. The key is REQUIRING_BUNDLES and the type is JmxConstants.LONG_ARRAY_TYPE

public static final String RESOLVED = "RESOLVED"
Constant RESOLVED for the STATE

public static final String SERVICES_IN_USE = "ServicesInUse"
The key SERVICES_IN_USE, used in SERVICES_IN_USE_ITEM.

public static final Item SERVICES_IN_USE_ITEM
The item containing the services in use by this bundle in BUNDLE_TYPE. The key is SERVICES_IN_USE and the the type is JmxConstants.LONG_ARRAY_TYPE

public static final String START_LEVEL = "StartLevel"
The key START_LEVEL, used in START_LEVEL_ITEM.

public static final Item START_LEVEL_ITEM
The item containing the start level in BUNDLE_TYPE. The key is START_LEVEL and the the type is SimpleType.INTEGER.

public static final String STARTING = "STARTING"
Constant STARTING for the STATE

public static final String STATE = "State"
The key STATE, used in STATE_ITEM.
124.8.2.52  public static final Item STATE_ITEM

The item containing the bundle state in BUNDLE_TYPE. The key is STATE and the the type is SimpleType.STRING. The returned values must be one of the following strings:

- INSTALLED
- RESOLVED
- STARTING
- ACTIVE
- STOPPING
- UNINSTALLED
- UNKNOWN

124.8.2.53  public static final String STOPPING = "STOPPING"

Constant STOPPING for the STATE

124.8.2.54  public static final String SYMBOLIC_NAME = "SymbolicName"

The key SYMBOLIC_NAME, used in SYMBOLIC_NAME_ITEM.

124.8.2.55  public static final Item SYMBOLIC_NAME_ITEM

The item containing the symbolic name in BUNDLE_TYPE. The key is SYMBOLIC_NAME and the the type is SimpleType.STRING.

124.8.2.56  public static final String UNINSTALLED = "UNINSTALLED"

Constant UNINSTALLED for the STATE

124.8.2.57  public static final String UNKNOWN = "UNKNOWN"

Constant UNKNOWN for the STATE

124.8.2.58  public static final String VALUE = "Value"

The key VALUE, used in VALUE_ITEM.

124.8.2.59  public static final Item VALUE_ITEM

The item describing the value of a bundle header entry. The key is VALUE and the type is SimpleType.STRING.

124.8.2.60  public static final String VERSION = "Version"

The key VERSION, used in VERSION_ITEM.

124.8.2.61  public static final Item VERSION_ITEM

The item containing the symbolic name in BUNDLE_TYPE. The key is SYMBOLIC_NAME and the the type is SimpleType.STRING.

124.8.2.62  public CompositeData getBundle(long bundleIdentifier) throws IOException

.bundleIdentifier  the bundle identifier of the requested bundle

Obtain the information regarding a single bundle. The result is defined by the BUNDLE_TYPE CompositeType.

Returns  A CompositeData object with the bundle information

Throws  IOException – if the operation fails
124.8.2.63  public long[] getBundleIds() throws IOException

- List all bundle IDs in the framework.

**Returns**  all the bundle ids in the framework.

**Throws**  IOException— if the operation fails

124.8.2.64  public String[] getExportedPackages(long bundleId) throws IOException

- Answer the list of exported packages for this bundle.

**Returns**  the array of package names, combined with their version in the format <packageName;version>

**Throws**  IOException— if the operation fails

- IllegalArgumentException— if the bundle indicated does not exist

124.8.2.65  public long[] getFragments(long bundleId) throws IOException

- Answer the list of the bundle ids of the fragments associated with this bundle

**Returns**  the array of bundle identifiers

**Throws**  IOException— if the operation fails

- IllegalArgumentException— if the bundle indicated does not exist

124.8.2.66  public String getHeader(long bundleId, String key) throws IOException

- the unique identifier of the bundle

- the key of the header to look up

- Retrieve a single header from the bundle headers.

**Returns**  the value of associated header

**Throws**  IOException— if the operation fails

- IllegalArgumentException— if the bundle indicated does not exist

124.8.2.67  public String getHeader(long bundleId, String key, String locale) throws IOException

- the unique identifier of the bundle

- the key of the header to look up

- the locale name into which the header value is to be localized. The value of this parameter follows the same rules as the locale parameter in Bundle.getHeaders(String locale)

- Retrieve a single header from the bundle headers.

- This method performs the same function as getHeaders(long bundleId) except the manifest header values are localized to the specified locale.

**Returns**  the value of associated header

**Throws**  IOException— if the operation fails

- IllegalArgumentException— if the bundle indicated does not exist

124.8.2.68  public TabularData getHeaders(long bundleId) throws IOException

- the unique identifier of the bundle
Answer the headers for the bundle uniquely identified by the bundle id. The Tabular Data is typed by the HEADERS_TYPE.

Returns the table of associated header key and values

Throws IOException – if the operation fails

IllegalArgumentException – if the bundle indicated does not exist

**124.8.2.69**

```java
public TabularData getHeaders(long bundleId, String locale) throws IOException
```

**bundleId**
the unique identifier of the bundle

**locale**
the locale name into which the header values are to be localized. The value of this parameter follows the same rules as the locale parameter in `Bundle.getHeaders(String locale)`

Answer the headers for the bundle uniquely identified by the bundle id. The Tabular Data is typed by the HEADERS_TYPE.

This method performs the same function as `getHeaders(long bundleId)` except the manifest header values are localized to the specified locale.

Returns the table of associated header key and values

Throws IOException – if the operation fails

IllegalArgumentException – if the bundle indicated does not exist

**124.8.2.70**

```java
public long[] getHosts(long fragment) throws IOException
```

**fragment**
the bundle id of the fragment

Answer the list of bundle ids of the bundles which host a fragment

Returns the array of bundle identifiers

Throws IOException – if the operation fails

IllegalArgumentException – if the bundle indicated does not exist

**124.8.2.71**

```java
public String[] getImportedPackages(long bundleId) throws IOException
```

**bundleId**
the bundle identifier

Answer the array of the packages imported by this bundle

Returns the array of package names, combined with their version in the format `<packageName;version>`

Throws IOException – if the operation fails

IllegalArgumentException – if the bundle indicated does not exist

**124.8.2.72**

```java
public long getLastModified(long bundleId) throws IOException
```

**bundleId**
the unique identifier of a bundle

Answer the last modified time of a bundle

Returns the last modified time

Throws IOException – if the operation fails

IllegalArgumentException – if the bundle indicated does not exist

**124.8.2.73**

```java
public String getLocation(long bundleId) throws IOException
```

**bundleId**
the identifier of the bundle

Answer the location of the bundle.

Returns The location string of this bundle
124.8.2.74  public long[] getRegisteredServices(long bundleId) throws IOException

   bundleId  the bundle identifier

   □  Answer the list of service identifiers representing the services this bundle exports

   Returns  the list of service identifiers

   Throws  IOException – if the operation fails

        IllegalArgumentException – if the bundle indicated does not exist

124.8.2.75  public long[] getRequiredBundles(long bundleIdentifier) throws IOException

   bundleIdentifier  the bundle identifier to find the dependencies for

   □  Answer an array of ids of bundles the given bundle depends on.

   Returns  the bundle identifiers of the dependencies

   Throws  IOException – if the operation fails

        IllegalArgumentException – if the bundle indicated does not exist

124.8.2.76  public long[] getRequiringBundles(long bundleIdentifier) throws IOException

   bundleIdentifier  the bundle identifier

   □  Answer the list of identifiers of the bundles which require this bundle

   Returns  the list of bundle identifiers

   Throws  IOException – if the operation fails

        IllegalArgumentException – if the bundle indicated does not exist

124.8.2.77  public long[] getServicesInUse(long bundleIdentifier) throws IOException

   bundleIdentifier  the bundle identifier

   □  Answer the list of service identifiers which refer to the the services this bundle is using

   Returns  the list of service identifiers

   Throws  IOException – if the operation fails

        IllegalArgumentException – if the bundle indicated does not exist

124.8.2.78  public int getStartLevel(long bundleId) throws IOException

   bundleId  the identifier of the bundle

   □  Answer the start level of the bundle

   Returns  the start level

   Throws  IOException – if the operation fails

        IllegalArgumentException – if the bundle indicated does not exist

124.8.2.79  public String getState(long bundleId) throws IOException

   bundleId  the identifier of the bundle

   □  Answer the symbolic name of the state of the bundle

   Returns  the string name of the bundle state

   Throws  IOException – if the operation fails
public String getSymbolicName(long bundleId) throws IOException

*bundleId* the identifier of the bundle

- Answer the symbolic name of the bundle

**Returns** the symbolic name

**Throws** IOException – if the operation fails

IllegalArgumentException – if the bundle indicated does not exist

public String getVersion(long bundleId) throws IOException

*bundleId* the identifier of the bundle

- Answer the location of the bundle.

**Returns** The location string of this bundle

**Throws** IOException – if the operation fails

IllegalArgumentException – if the bundle indicated does not exist

public boolean isActivationPolicyUsed(long bundleId) throws IOException

*bundleId* the identifier of the bundle

- Answer whether the specified bundle's autostart setting indicates that the activation policy declared in the bundle's manifest must be used.

**Returns** true if the bundle's autostart setting indicates the activation policy declared in the manifest must be used. false if the bundle must be eagerly activated.

**Throws** IOException – if the operation fails

IllegalArgumentException – if the bundle indicated does not exist

public boolean isFragment(long bundleId) throws IOException

*bundleId* the identifier of the bundle

- Answer whether the bundle is a fragment or not

**Returns** true if the bundle is a fragment

**Throws** IOException – if the operation fails

IllegalArgumentException – if the bundle indicated does not exist

public boolean isPersistentlyStarted(long bundleId) throws IOException

*bundleId* the identifier of the bundle

- Answer if the bundle is persistently started when its start level is reached

**Returns** true if the bundle is persistently started

**Throws** IOException – if the operation fails

IllegalArgumentException – if the bundle indicated does not exist

public boolean isRemovalPending(long bundleId) throws IOException

*bundleId* the identifier of the bundle

- Answer true if the bundle is pending removal

**Returns** true if the bundle is pending removal
124.8.2.86 public boolean isRequired(long bundleId) throws IOException

*bundleId* the identifier of the bundle

- Answer true if the bundle is required by another bundle

*Returns* true if the bundle is required by another bundle

*Throws* IOException – if the operation fails

124.8.2.87 public TabularData listBundles() throws IOException

- Answer the bundle state of the system in tabular form. Each row of the returned table represents a single bundle. The Tabular Data consists of Composite Data that is type by BUNDLES_TYPE.

*Returns* the tabular representation of the bundle state

*Throws* IOException – if the operation fails

124.8.2.88 public TabularData listBundles(String ... items) throws IOException

*items* The names of the items to include in the result.

- Answer the bundle state of the system in tabular form. Each row of the returned table represents a single bundle. The Tabular Data consists of Composite Data that is type by BUNDLES_TYPE. This method supports specifying the items that are included in the result. Note that the IDENTIFIER item is always returns as this the key in the TabularData structure.

*Returns* the tabular representation of the bundle state

*Throws* IOException – if the operation fails

124.8.3 public interface FrameworkMBean

The FrameworkMbean provides mechanisms to exert control over the framework. For many operations, it provides a batch mechanism to avoid excessive message passing when interacting remotely.

*Concurrency* Thread-safe

124.8.3.1 public static final CompositeType BATCH_ACTION_RESULT_TYPE

The Composite Type for a batch action result. refreshBundle(long) and refreshBundles(long[]). Notice that a batch action result returns uses an id for the BUNDLE_IN_ERROR while the BATCH_INSTALL_RESULT_TYPE uses a location. This Composite Type consists of the following items:

- BUNDLE_IN_ERROR_ID_ITEM
- COMPLETED_ITEM
- ERROR_ITEM
- REMAINING_ID_ITEM
- SUCCESS_ITEM

124.8.3.2 public static final CompositeType BATCH_INSTALL_RESULT_TYPE

The Composite Type which represents the result of a batch install operation. It is used in installBundles(String[]) and installBundlesFromURL(String[], String[]). This Composite Type consists of the following items:

- BUNDLE_IN_ERROR_LOCATION_ITEM
• COMPLETED_ITEM
• ERROR_ITEM
• REMAINING_LOCATION_ITEM
• SUCCESS_ITEM

124.8.3.3 public static final CompositeType BATCH_RESOLVE_RESULT_TYPE
The Composite Type which represents the result of a batch resolve operation. It is used in
refreshBundlesAndWait(long[]) and resolve(long[]). This Composite Type consists of the following
items:
• COMPLETED_ITEM
• SUCCESS_ITEM

124.8.3.4 public static final String BUNDLE_IN_ERROR = "BundleInError"
The key for BUNDLE_IN_ERROR. This key is used with two different items:
BUNDLE_IN_ERROR_ID_ITEM and BUNDLE_IN_ERROR_LOCATION_ITEM that each
have a different type for this key. It is used in BATCH_ACTION_RESULT_TYPE and
BATCH_INSTALL_RESULT_TYPE.

124.8.3.5 public static final Item BUNDLE_IN_ERROR_ID_ITEM
The item containing the bundle which caused the error during the batch operation. This
item describes the bundle in error as an id. The key is BUNDLE_IN_ERROR and the type is
SimpleType.LONG. It is used in BATCH_ACTION_RESULT_TYPE.
See Also BUNDLE_IN_ERROR_LOCATION_ITEM for the item that has a location for the bundle in error.

124.8.3.6 public static final Item BUNDLE_IN_ERROR_LOCATION_ITEM
The item containing the bundle which caused the error during the batch operation. This item
describes the bundle in error as a location. The key is BUNDLE_IN_ERROR and the type is
SimpleType.STRING. It is used in BATCH_INSTALL_RESULT_TYPE.
See Also BUNDLE_IN_ERROR_ID_ITEM for the item that has the id for the bundle in error.

124.8.3.7 public static final String COMPLETED = "Completed"
The key COMPLETED, used in COMPLETED_ITEM.

124.8.3.8 public static final Item COMPLETED_ITEM
The item containing the list of bundles completing the batch operation. The key is COMPLETED
and the type is JmxConstants.LONG_ARRAY_TYPE. It is used in BATCH_ACTION_RESULT_TYPE
and BATCH_INSTALL_RESULT_TYPE.

124.8.3.9 public static final String ERROR = "Error"
The key ERROR, used in ERROR_ITEM.

124.8.3.10 public static final Item ERROR_ITEM
The item containing the error message of the batch operation. The key is ERROR and
the type is SimpleType.STRING. It is used in BATCH_ACTION_RESULT_TYPE and
BATCH_INSTALL_RESULT_TYPE.

124.8.3.11 public static final String OBJECTNAME = "osgi.core:type=framework,version=1.7"
The Object Name prefix for this mbean. The full object name also contains the framework name and
uuid as properties.
124.8.3.12 public static final String REMAINING = "Remaining"

The key REMAINING, used in REMAINING_ID_ITEM and REMAINING_LOCATION_ITEM.

124.8.3.13 public static final Item REMAINING_ID_ITEM

The item containing the list of remaining bundles unprocessed by the failing batch operation. The key is REMAINING and the type is JmxConstants.LONG_ARRAY_TYPE. It is used in BATCH_ACTION_RESULT_TYPE and BATCH_INSTALL_RESULT_TYPE.

124.8.3.14 public static final Item REMAINING_LOCATION_ITEM

The item containing the list of remaining bundles unprocessed by the failing batch operation. The key is REMAINING and the type is JmxConstants.STRING_ARRAY_TYPE. It is used in BATCH_ACTION_RESULT_TYPE and BATCH_INSTALL_RESULT_TYPE.

124.8.3.15 public static final String SUCCESS = "Success"

The SUCCESS, used in SUCCESS_ITEM.

124.8.3.16 public static final Item SUCCESS_ITEM

The item that indicates if this operation was successful. The key is SUCCESS and the type is SimpleType.BOOLEAN. It is used in BATCH_ACTION_RESULT_TYPE and BATCH_INSTALL_RESULT_TYPE.

124.8.3.17 public long[] getDependencyClosure(long[] bundles) throws IOException

bundles The initial bundles IDs for which to generate the dependency closure.

Returns the dependency closure for the specified bundles.

A graph of bundles is computed starting with the specified bundles. The graph is expanded by adding any bundle that is either wired to a package that is currently exported by a bundle in the graph or requires a bundle in the graph. The graph is fully constructed when there is no bundle outside the graph that is wired to a bundle in the graph. The graph may contain UNINSTALLED bundles that are removal pending.

Returns A bundle ID array containing a snapshot of the dependency closure of the specified bundles, or an empty array if there were no specified bundles.

Throws IOException – if the operation failed

IllegalArgumentException – if a bundle indicated does not exist

124.8.3.18 public int getFrameworkStartLevel() throws IOException

Returns the framework start level

Throws IOException – if the operation failed

124.8.3.19 public int getInitialBundleStartLevel() throws IOException

Returns the start level

Throws IOException – if the operation failed

124.8.3.20 public String getProperty(String key) throws IOException

key The name of the requested property.

Returns the value of the specified property. If the key is not found in the Framework properties, the system properties are then searched. The method returns null if the property is not found.
Returns  The value of the requested property, or null if the property is undefined.

Throws  IOException – if the operation failed

124.8.3.21  public long[] getRemovalPendingBundles() throws IOException

Returns  The bundles IDs that have non-current, in use bundle wirings. This is typically the bundles which have been updated or uninstalled since the last call to refreshBundles(long[]).

Throws  IOException – if the operation failed

124.8.3.22  public long installBundle(String location) throws IOException

location  the location of the bundle to install

Returns  the bundle ID of the installed bundle

Throws  IOException – if the operation does not succeed

124.8.3.23  public long installBundleFromURL(String location, String url) throws IOException

location  the location to assign to the bundle

url  the URL which will supply the bytes for the bundle

Returns  the bundle ID of the installed bundle

Throws  IOException – if the operation does not succeed

124.8.3.24  public CompositeData installBundles(String[] locations) throws IOException

locations  the array of locations of the bundles to install

Returns  the resulting state from executing the operation

Throws  IOException – if the operation does not succeed

See Also  BATCH_INSTALL_RESULT_TYPE for the precise specification of the CompositeData type representing the returned result.

124.8.3.25  public CompositeData installBundlesFromURL(String[] locations, String[] urls) throws IOException

locations  the array of locations to assign to the installed bundles

urls  the array of urls which supply the bundle bytes

Returns  the resulting state from executing the operation

Throws  IOException – if the operation does not succeed

See Also  for the precise specification of the CompositeData type representing the returned result.

124.8.3.26  public void refreshBundle(long bundleIdentifier) throws IOException

bundleIdentifier  the bundle identifier

Returns  The update, replacement or removal of the packages identified by the specified bundle.

Throws  IOException – if the operation failed

IllegalArgumentException – if the bundle indicated does not exist
### 124.8.3.27 refreshBundleAndWait

<table>
<thead>
<tr>
<th>public boolean refreshBundleAndWait(long bundleIdentifier) throws IOException</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bundleIdentifier</strong></td>
</tr>
<tr>
<td>□</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td><strong>Throws</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

### 124.8.3.28 refreshBundles

<table>
<thead>
<tr>
<th>public void refreshBundles(long[] bundleIdentifiers) throws IOException</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bundleIdentifiers</strong></td>
</tr>
<tr>
<td>□</td>
</tr>
<tr>
<td><strong>Throws</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

### 124.8.3.29 refreshBundlesAndWait

<table>
<thead>
<tr>
<th>public CompositeData refreshBundlesAndWait(long[] bundleIdentifiers) throws IOException</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bundleIdentifiers</strong></td>
</tr>
<tr>
<td>□</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td><strong>Throws</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>See Also</strong></td>
</tr>
</tbody>
</table>

### 124.8.3.30 resolve

<table>
<thead>
<tr>
<th>public CompositeData resolve(long[] bundleIdentifiers) throws IOException</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bundleIdentifiers</strong></td>
</tr>
<tr>
<td>□</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td><strong>Throws</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>See Also</strong></td>
</tr>
</tbody>
</table>

### 124.8.3.31 resolveBundle

<table>
<thead>
<tr>
<th>public boolean resolveBundle(long bundleIdentifier) throws IOException</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bundleIdentifier</strong></td>
</tr>
<tr>
<td>□</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td><strong>Throws</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

### 124.8.3.32 resolveBundles

<table>
<thead>
<tr>
<th>public boolean resolveBundles(long[] bundleIdentifiers) throws IOException</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bundleIdentifiers</strong></td>
</tr>
<tr>
<td>□</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td><strong>Throws</strong></td>
</tr>
</tbody>
</table>
124.8.3.33 public void restartFramework() throws IOException

- Restart the framework by updating the system bundle

Throws IOException – if the operation failed

124.8.3.34 public void setBundleStartLevel(long bundleIdentifier, int newLevel) throws IOException

- BundleIdentifier the bundle identifier
- newLevel the new start level for the bundle

Set the start level for the bundle identifier

Throws IOException – if the operation failed

124.8.3.35 public CompositeData setBundleStartLevels(long[] bundleIdentifiers, int[] newlevels) throws IOException

- bundleIdentifiers the array of bundle identifiers
- newlevels the array of new start level for the bundles

Set the start levels for the list of bundles.

Returns the resulting state from executing the operation

Throws IOException – if the operation failed

See Also for the precise specification of the CompositeData type representing the returned result.

124.8.3.36 public void setFrameworkStartLevel(int newlevel) throws IOException

- newLevel the new start level

Set the start level for the framework

Throws IOException – if the operation failed

124.8.3.37 public void setInitialBundleStartLevel(int newlevel) throws IOException

- newLevel the new start level

Set the initial start level assigned to a bundle when it is first started

Throws IOException – if the operation failed

124.8.3.38 public void shutdownFramework() throws IOException

- Shutdown the framework by stopping the system bundle

Throws IOException – if the operation failed

124.8.3.39 public void startBundle(long bundleIdentifier) throws IOException

- BundleIdentifier the bundle identifier

Start the bundle indicated by the bundle identifier

Throws IOException – if the operation does not succeed

IllegalArgumentException – if the bundle indicated does not exist

124.8.3.40 public CompositeData startBundles(long[] bundleIdentifiers) throws IOException

- bundleIdentifiers the array of bundle identifiers

Batch start the bundles indicated by the list of bundle identifier

Returns the resulting state from executing the operation
124.8.3.41 public void stopBundle(long bundleIdentifier) throws IOException

bundleIdentifier the bundle identifier

■ Stop the bundle indicated by the bundle identifier

Throws IOException – if the operation does not succeed

IllegalArgumentException – if the bundle indicated does not exist

See Also for the precise specification of the CompositeData type representing the returned result.

124.8.3.42 public CompositeData stopBundles(long[] bundleIdentifiers) throws IOException

bundleIdentifiers the array of bundle identifiers

■ Batch stop the bundles indicated by the list of bundle identifier

Returns the resulting state from executing the operation

Throws IOException – if the operation does not succeed

See Also BATCH_ACTION_RESULT_TYPE for the precise specification of the CompositeData type representing the returned result.

124.8.3.43 public void uninstallBundle(long bundleIdentifier) throws IOException

bundleIdentifier the bundle identifier

■ Uninstall the bundle indicated by the bundle identifier

Throws IOException – if the operation does not succeed

IllegalArgumentException – if the bundle indicated does not exist

124.8.3.44 public CompositeData uninstallBundles(long[] bundleIdentifiers) throws IOException

bundleIdentifiers the array of bundle identifiers

■ Batch uninstall the bundles indicated by the list of bundle identifiers

Returns the resulting state from executing the operation

Throws IOException – if the operation does not succeed

See Also BATCH_ACTION_RESULT_TYPE for the precise specification of the CompositeData type representing the returned result.

124.8.3.45 public void updateBundle(long bundleIdentifier) throws IOException

bundleIdentifier the bundle identifier

■ Update the bundle indicated by the bundle identifier

Throws IOException – if the operation does not succeed

IllegalArgumentException – if the bundle indicated does not exist

124.8.3.46 public void updateBundleFromURL(long bundleIdentifier,String url) throws IOException

bundleIdentifier the bundle identifier

url the URL to use to update the bundle

■ Update the bundle identified by the bundle identifier

Throws IOException – if the operation does not succeed

IllegalArgumentException – if the bundle indicated does not exist
124.8.3.47  public CompositeData updateBundles(long[] bundleIdentifiers) throws IOException

**bundleIdentifiers** the array of bundle identifiers

- Batch update the bundles indicated by the list of bundle identifier.

**Returns** the resulting state from executing the operation

**Throws** IOException -- if the operation does not succeed

**See Also** BATCH_ACTION_RESULT_TYPE for the precise specification of the CompositeData type representing the returned result.

124.8.3.48  public CompositeData updateBundlesFromURL(long[] bundleIdentifiers, String[] urls) throws IOException

**bundleIdentifiers** the array of bundle identifiers

**urls** the array of URLs to use to update the bundles

- Update the bundle uniquely identified by the bundle symbolic name and version using the contents of the supplied urls.

**Returns** the resulting state from executing the operation

**Throws** IOException -- if the operation does not succeed

IllegalArgumentException -- if the bundle indicated does not exist

**See Also** BATCH_ACTION_RESULT_TYPE for the precise specification of the CompositeData type representing the returned result.

124.8.3.49  public void updateFramework() throws IOException

- Update the framework by updating the system bundle.

**Throws** IOException -- if the operation failed

124.8.4  public interface PackageStateMBean

This MBean provides information about the package state of the framework.

**Concurrency** Thread-safe

124.8.4.1  public static final String EXPORTING_BUNDLES = "ExportingBundles"

The key EXPORTING_BUNDLE, used in EXPORTING_BUNDLES_ITEM.

124.8.4.2  public static final Item EXPORTING_BUNDLES_ITEM

The item containing the bundle identifier in PACKAGE_TYPE. The key is EXPORTING_BUNDLES and the type is JmxConstants.LONG_ARRAY_TYPE.

124.8.4.3  public static final String IMPORTING_BUNDLES = "ImportingBundles"

The key IMPORTING_BUNDLES, used in IMPORTING_BUNDLES_ITEM.

124.8.4.4  public static final Item IMPORTING_BUNDLES_ITEM

The item containing the bundle identifier in PACKAGE_TYPE. The key is IMPORTING_BUNDLES and the type is JmxConstants.LONG_ARRAY_TYPE.

124.8.4.5  public static final String NAME = "Name"

The key NAME, used in NAME_ITEM.

124.8.4.6  public static final Item NAME_ITEM

The item containing the name of the package in PACKAGE_TYPE. The key is NAME and the type is SimpleType.LONG.
124.8.4.7 public static final String OBJECTNAME = "osgi.core:type=packageState,version=1.5"
The fully qualified object name of this MBean.

124.8.4.8 public static final CompositeType PACKAGE_TYPE
The Composite Type for a CompositeData representing a package. This type consists of:

- EXPORTING_BUNDLES_ITEM
- IMPORTING_BUNDLES_ITEM
- NAME_ITEM
- REMOVAL_PENDING_ITEM
- VERSION_ITEM

The key is defined as NAME and EXPORTING_BUNDLES

124.8.4.9 public static final TabularType PACKAGES_TYPE
The Tabular Type used in listPackages(). They key is NAME, VERSION, and EXPORTING_BUNDLES.

124.8.4.10 public static final String REMOVAL_PENDING = "RemovalPending"
The name of the item containing the pending removal status of the package in the CompositeData. Used

124.8.4.11 public static final Item REMOVAL_PENDING_ITEM
The item representing the removal pending status of a package. The key is REMOVAL_PENDING and the type is SimpleType.BOOLEAN.

124.8.4.12 public static final String VERSION = "Version"
The name of the item containing the package version in the CompositeData. Used in VERSION_ITEM.

124.8.4.13 public static final Item VERSION_ITEM
The item containing the version of the package in PACKAGE_TYPE. The key is VERSION and the type is SimpleType.STRING.

124.8.4.14 public long[] getExportingBundles(String packageName, String version) throws IOException

- packageName - the package name
- version - the version of the package

□ Answer the identifier of the bundle exporting the package

Returns the bundle identifiers exporting such a package

Throws IOException – if the operation fails

IllegalArgumentException – if the package indicated does not exist

124.8.4.15 public long[] getImportingBundles(String packageName, String version, long exportingBundle) throws IOException

- packageName The package name
- version The version of the package
- exportingBundle The exporting bundle for the given package

□ Answer the list of identifiers of the bundles importing the package

Returns the list of bundle identifiers
124.8.4.16 public boolean isRemovalPending(String packageName, String version, long exportingBundle) throws IOException

**packageName** The package name

**version** The version of the package

**exportingBundle** The bundle exporting the package

- Answer if this package is exported by a bundle which has been updated or uninstalled

**Returns** true if this package is being exported by a bundle that has been updated or uninstalled.

**Throws** IOException – if the operation fails

IllegalArgumentException – if the package indicated does not exist

124.8.4.17 public TabularData listPackages() throws IOException

- Answer the package state of the system in tabular form The Tabular Data is typed by PACKAGES_TYPE, which has PACKAGE_TYPE as its Composite Type.

**Returns** the tabular representation of the package state

**Throws** IOException – When fails

124.8.5 public interface ServiceStateMBean

This MBean represents the Service state of the framework. This MBean also emits events that clients can use to get notified of the changes in the service state of the framework.

**Concurrency** Thread-safe

124.8.5.1 public static final String BUNDLE_IDENTIFIER = "BundleIdentifier"

The key BUNDLE_IDENTIFIER, used in BUNDLE_IDENTIFIER_ITEM.

124.8.5.2 public static final Item BUNDLE_IDENTIFIER_ITEM

The item containing the bundle identifier in SERVICE_TYPE. The key is BUNDLE_IDENTIFIER and the type is SimpleType.LONG.

124.8.5.3 public static final String BUNDLE_LOCATION = "BundleLocation"

The key BUNDLE_LOCATION, used in SERVICE_EVENT_TYPE.

124.8.5.4 public static final Item BUNDLE_LOCATION_ITEM

The item containing the bundle location in EVENT_ITEM. The key is BUNDLE_LOCATION and the type is SimpleType.STRING.

124.8.5.5 public static final String BUNDLE_SYMBOLIC_NAME = "BundleSymbolicName"

The key BUNDLE_SYMBOLIC_NAME, used in SERVICE_EVENT_TYPE.

124.8.5.6 public static final Item BUNDLE_SYMBOLIC_NAME_ITEM

The item containing the symbolic name in EVENT. The key is BUNDLE_SYMBOLIC_NAME and the type is SimpleType.STRING.

124.8.5.7 public static final String EVENT = "ServiceEvent"

The key EVENT, used in EVENT_ITEM.
124.8.5.8 public static final Item EVENT_ITEM
   The item containing the event type. The key is EVENT and the type is SimpleType.INTEGER

124.8.5.9 public static final String IDENTIFIER = "Identifier"
   The key IDENTIFIER, used IDENTIFIER_ITEM.

124.8.5.10 public static final Item IDENTIFIER_ITEM
   The item containing the service identifier in SERVICE_TYPE. The key is IDENTIFIER and the type is SimpleType.LONG.

124.8.5.11 public static final String OBJECT_CLASS = "objectClass"
   The key OBJECT_CLASS, used OBJECT_CLASS_ITEM.

124.8.5.12 public static final Item OBJECT_CLASS_ITEM
   The item containing the interfaces of the service in SERVICE_TYPE. The key is OBJECT_CLASS and the type is JmxConstants.STRING_ARRAY_TYPE.

124.8.5.13 public static final String OBJECTNAME = "osgi.core:type=serviceState,version=1.7"
   The Object Name prefix for this mbean. The full object name also contains the framework name and uuid as properties.

124.8.5.14 public static final String PROPERTIES = "Properties"
   The key PROPERTIES, used in PROPERTIES_ITEM.

124.8.5.15 public static final Item PROPERTIES_ITEM
   The item containing service properties in SERVICE_TYPE. The key is PROPERTIES and the type is JmxConstants.PROPERTIES_TYPE.

124.8.5.16 public static final CompositeType SERVICE_EVENT_TYPE
   The Composite Type that represents a service event. This composite consists of:
   
   - IDENTIFIER
   - OBJECT_CLASS
   - BUNDLE_LOCATION
   - BUNDLE_SYMBOLIC_NAME
   - EVENT

124.8.5.17 public static final CompositeType SERVICE_TYPE
   The Composite Type for a CompositeData representing a service. This type consists of:
   
   - BUNDLE_IDENTIFIER
   - IDENTIFIER
   - OBJECT_CLASS
   - PROPERTIES
   - USING_BUNDLES

124.8.5.18 public static final TabularType SERVICES_TYPE
   The Tabular Type for a Service table. The rows consists of SERVICE_TYPE Composite Data and the index is IDENTIFIER.
124.8.5.19  public static final String USING_BUNDLES = "UsingBundles"

The key USING_BUNDLES, used in USING_BUNDLES_ITEM.

124.8.5.20  public static final Item USING_BUNDLES_ITEM

The item containing the bundles using the service in SERVICE_TYPE. The key is USING_BUNDLES
and the type is JmxConstants.LONG_ARRAY_TYPE.

124.8.5.21  public long getBundleIdentifier(long serviceId) throws IOException

ServiceId   the identifier of the service

□ Answer the bundle identifier of the bundle which registered the service

Returns   the identifier for the bundle

Throws   IOException – if the operation fails

IllegalArgumentException – if the service indicated does not exist

124.8.5.22  public String[] getobjectClass(long serviceId) throws IOException

ServiceId   the identifier of the service

□ Answer the list of interfaces that this service implements

Returns   the list of interfaces

Throws   IOException – if the operation fails

IllegalArgumentException – if the service indicated does not exist

124.8.5.23  public TabularData getProperties(long serviceId) throws IOException

ServiceId   the identifier of the service

□ Answer the map of properties associated with this service.

Returns   the table of properties. These include the standard mandatory service.id and objectClass properties
as defined in the org.osgi.framework.Constants interface

Throws   IOException – if the operation fails

IllegalArgumentException – if the service indicated does not exist

See Also   for the details of the TabularType

124.8.5.24  public CompositeData getProperty(long serviceId,String key) throws IOException

ServiceId   the identifier of the service

key   the property key

□ Return a single property from the specified service.

Returns   a CompositeData object holding the value and data type of the property.

Throws   IOException – if the operation fails

See Also   for the details of the CompositeType.

124.8.5.25  public CompositeData getService(long serviceId) throws IOException

ServiceId   the ID of the service to look up

□ Obtain information about a given service. The result is defined by the CompositeType

SERVICE_TYPE.

Returns   A CompositeData object with the service information

Throws   IOException – if the operation fails
IllegalArgumentException– If the service indicated does not exist

124.8.5.26  public long[] getServiceIds() throws IOException
  [ ] List all service IDs in the framework.
  Returns all the service ids in the framework.
  Throws IOException– if the operation fails

124.8.5.27  public long[] getUsingBundles(long serviceId) throws IOException
  serviceId the identifier of the service
  [ ] Answer the list of identifiers of the bundles that use the service
  Returns the list of bundle identifiers
  Throws IOException– if the operation fails
  IllegalArgumentException– if the service indicated does not exist

124.8.5.28  public TabularData listServices() throws IOException
  [ ] Answer the service state of the system in tabular form.
  Returns the tabular representation of the service state
  Throws IOException– if the operation fails
  IllegalArgumentException– if the service indicated does not exist
  See Also for the details of the TabularType

124.8.5.29  public TabularData listServices(String clazz, String filter) throws IOException
  clazz The class name with which the services were registered or null for any class name.
  filter A filter expression to match the services or null for no additional filter.
  [ ] Answer the service state of the system in tabular form. This method allows the specification of a
    class name and a filter to select services to be provided.
  Returns the tabular representation of the service state
  Throws IOException– if the operation fails
  IllegalArgumentException– if the service indicated does not exist
  See Also for the details of the TabularType

124.8.5.30  public TabularData listServices(String clazz, String filter, String ... serviceTypeItems) throws IOException
  clazz The class name with which the services were registered or null for any class name.
  filter A filter expression to match the services or null for no additional filter.
  serviceTypeItems The names of the SERVICE_TYPE items to include in the result. For example "objectClass" or "Properties". Note that the result always returns the "Identifier" item since this serves as the key in the resulting table.
  [ ] Answer the service state of the system in tabular form. Apart from class name and filter, this method allows the specification of a subset of the SERVICE_TYPE items to be included in the result. Selecting only the relevant Service Type items may save bandwidth and improve performance over a remote connection.
  Returns the tabular representation of the service state
  Throws IOException– if the operation fails
  IllegalArgumentException– if the service indicated does not exist
OSGi JMX Configuration Admin Package Version 1.3.

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.jmx.service.cm; version="[1.3,2.0)"
```

Example import for providers implementing the API in this package:
```
Import-Package: org.osgi.jmx.service.cm; version="[1.3,1.4)"
```

**124.9.1 public interface ConfigurationAdminMBean**

This MBean provides the management interface to the OSGi Configuration Administration Service.

*Concurrency* Thread-safe

**124.9.1.1 public static final String OBJECTNAME = "osgi.compendium:service=cm,version=1.3"**

The object name for this mbean.

**124.9.1.2 public String createFactoryConfiguration(String factoryPid) throws IOException**

*factoryPid* the persistent id of the factory

- Create a new configuration instance for the supplied persistent id of the factory, answering the PID of the created configuration

*Returns* the PID of the created configuration

*Throws* IOException – if the operation failed

**124.9.1.3 public String createFactoryConfigurationForLocation(String factoryPid, String location) throws IOException**

*factoryPid* the persistent id of the factory

*location* the bundle location

- Create a factory configuration for the supplied persistent id of the factory and the bundle location bound to bind the created configuration to, answering the PID of the created configuration

*Returns* the pid of the created configuration

*Throws* IOException – if the operation failed

**124.9.1.4 public void delete(String pid) throws IOException**

*pid* the persistent identifier of the configuration

- Delete the configuration

*Throws* IOException – if the operation fails

**124.9.1.5 public void deleteConfigurations(String filter) throws IOException**

*filter* the string representation of the `org.osgi.framework.Filter`

- Delete the configurations matching the filter specification.

*Throws* IOException – if the operation failed
IllegalArgumentException – if the filter is invalid

124.9.1.6 public void deleteForLocation(String pid, String location) throws IOException

   pid the persistent identifier of the configuration
   location the bundle location
   □ Delete the configuration
   Throws IOException – if the operation fails

124.9.1.7 public String getBundleLocation(String pid) throws IOException

   pid the persistent identifier of the configuration
   □ Answer the bundle location the configuration is bound to
   Returns the bundle location
   Throws IOException – if the operation fails

124.9.1.8 public String[][] getConfigurations(String filter) throws IOException

   filter the string representation of the org.osgi.framework.Filter
   □ Answer the list of PID/Location pairs of the configurations managed by this service
   Returns the list of configuration PID/Location pairs
   Throws IOException – if the operation failed
   IllegalArgumentException – if the filter is invalid

124.9.1.9 public String getFactoryPid(String pid) throws IOException

   pid the persistent identifier of the configuration
   □ Answer the factory PID if the configuration is a factory configuration, null otherwise.
   Returns the factory PID
   Throws IOException – if the operation fails

124.9.1.10 public String getFactoryPidForLocation(String pid, String location) throws IOException

   pid the persistent identifier of the configuration
   location the bundle location
   □ Answer the factory PID if the configuration is a factory configuration, null otherwise.
   Returns the factory PID
   Throws IOException – if the operation fails

124.9.1.11 public TabularData getProperties(String pid) throws IOException

   pid the persistent identifier of the configuration
   □ Answer the contents of the configuration.
   Returns the table of contents
   Throws IOException – if the operation fails
   See Also JmxConstants.PROPERTIES_TYPE for the details of the TabularType

124.9.1.12 public TabularData getPropertiesForLocation(String pid, String location) throws IOException

   pid the persistent identifier of the configuration
location the bundle location
  □ Answer the contents of the configuration.

Returns the table of contents

Throws IOException – if the operation fails

See Also JmxConstants.PROPERTIES_TYPE for the details of the TabularType

124.9.13 public void setBundleLocation(String pid, String location) throws IOException

pid the persistent identifier of the configuration
location the bundle location
  □ Set the bundle location the configuration is bound to

Throws IOException – if the operation fails

124.9.14 public void update(String pid, TabularData properties) throws IOException

pid the persistent identifier of the configuration
properties the table of properties
  □ Update the configuration with the supplied properties For each property entry, the following row is supplied.

Throws IOException – if the operation fails

See Also JmxConstants.PROPERTIES_TYPE for the details of the TabularType

124.9.15 public void updateForLocation(String pid, String location, TabularData properties) throws IOException

pid the persistent identifier of the configuration
location the bundle location
properties the table of properties
  □ Update the configuration with the supplied properties For each property entry, the following row is supplied.

Throws IOException – if the operation fails

See Also JmxConstants.PROPERTIES_TYPE for the details of the TabularType

124.10 org.osgi.jmx.service.permissionadmin

OSGi JMX Permission Admin Package Version 1.2.

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.jmx.service.permissionadmin; version="[1.2,2.0)"

Example import for providers implementing the API in this package:
Import-Package: org.osgi.jmx.service.permissionadmin; version="[1.2,1.3)"

124.10.1 public interface PermissionAdminMBean

This MBean represents the OSGi Permission Manager Service
Concurrency
Thread-safe

124.10.1.1 public static final String OBJECTNAME = "osgi.core:service=permissionadmin,version=1.2"
Permission Admin MBean object name.

124.10.1.2 public String[] getPermissions(String location) throws IOException
location
location identifying the bundle
Answer the list of encoded permissions of the bundle specified by the bundle location
Returns the array of String encoded permissions
Throws IOException -- if the operation fails

124.10.1.3 public String[] listDefaultPermissions() throws IOException
Answer the list of encoded permissions representing the default permissions assigned to bundle locations that have no assigned permissions
Returns the array of String encoded permissions
Throws IOException -- if the operation fails

124.10.1.4 public String[] listLocations() throws IOException
Answer the bundle locations that have permissions assigned to them
Returns the bundle locations
Throws IOException -- if the operation fails

124.10.1.5 public void setDefaultPermissions(String[] encodedPermissions) throws IOException
encodedPermissions
the string encoded permissions
Set the default permissions assigned to bundle locations that have no assigned permissions
Throws IOException -- if the operation fails

124.10.1.6 public void setPermissions(String location, String[] encodedPermissions) throws IOException
location
the location of the bundle
encodedPermissions
the string encoded permissions to set
Set the permissions on the bundle specified by the bundle location
Throws IOException -- if the operation fails

124.11 org.osgi.jmx.service.provisioning

OSGi JMX Initial Provisioning Package Version 1.2.
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.jmx.service.provisioning; version="[1.2,2.0)"

Example import for providers implementing the API in this package:
124.11.1  public interface ProvisioningServiceMBean

This MBean represents the management interface to the OSGi Initial Provisioning Service

Concurrency  Thread-safe

124.11.1.1  public static final String OBJECTNAME = "osgi.compendium:service=provisioning,version=1.2"

Provisioning MBean object name.

124.11.1.2  public void addInformation(TabularData info) throws IOException

info  the set of Provisioning Information key/value pairs to add to the Provisioning Information dictionary. Any keys are values that are of an invalid type will be silently ignored.

- Adds the key/value pairs contained in info to the Provisioning Information dictionary. This method causes the PROVISIONING_UPDATE_COUNT to be incremented.

Throws  IOException – if the operation fails

See Also  JmxConstants.PROPERTIES_TYPE for details of the Tabular Data

124.11.1.3  public void addInformationFromZip(String zipURL) throws IOException

zipURL  the String form of the URL that will be resolved into a ZipInputStream which will be used to add key/value pairs to the Provisioning Information dictionary and install start bundles. If a ZipEntry does not have an Extra field that corresponds to one of the four defined MIME types (MIME_STRING, MIME_BYTE_ARRAY, MIME_BUNDLE, and MIME_BUNDLE_URL) in will be silently ignored.

- Processes the ZipInputStream contents of the provided zipURL and extracts information to add to the Provisioning Information dictionary, as well as, install/update and start bundles. This method causes the PROVISIONING_UPDATE_COUNT to be incremented.

Throws  IOException – if an error occurs while processing the ZipInputStream of the URL. No additions will be made to the Provisioning Information dictionary and no bundles must be started or installed.

124.11.1.4  public TabularData listInformation() throws IOException

- Returns a table representing the Provisioning Information Dictionary.

Returns  The table representing the manager dictionary.

Throws  IOException – if the operation fails

See Also  JmxConstants.PROPERTIES_TYPE for details of the Tabular Data

124.11.1.5  public void setInformation(TabularData info) throws IOException

info  the new set of Provisioning Information key/value pairs. Any keys are values that are of an invalid type will be silently ignored.

- Replaces the Provisioning Information dictionary with the entries of the supplied table. This method causes the PROVISIONING_UPDATE_COUNT to be incremented.

Throws  IOException – if the operation fails

See Also  JmxConstants.PROPERTIES_TYPE for details of the Tabular Data

124.12  org.osgi.jmx.service.useradmin

OSGi JMX User Admin Package Version 1.1.
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 pack-
age and the providers that implement the API in this package.

Example import for consumers using the API in this package:
Import-Package: org.osgi.jmx.service.useradmin; version="[1.1,2.0)"

Example import for providers implementing the API in this package:
Import-Package: org.osgi.jmx.service.useradmin; version="[1.1,1.2)"

124.12.1 public interface UserAdminMBean

This MBean provides the management interface to the OSGi User Manager Service

Concurrence Thread-safe

124.12.1.1 public static final CompositeType AUTORIZATION_TYPE

The Composite Type for an Authorization object. It consists of the NAME_ITEM and ROLES_ITEM 
items.

124.12.1.2 public static final String CREDENTIALS = "Credentials"

The CREDENTIALS key, used in CREDENTIALS_ITEM.

124.12.1.3 public static final Item CREDENTIALS_ITEM

The item containing the credentials of a user. The key is CREDENTIALS and the type is 
JmxConstants.PROPERTIES_TYPE.

124.12.1.4 public static final CompositeType GROUP_TYPE

The Composite Type for a Group. It extends USER_TYPE and adds MEMBERS_ITEM, and 
REQUIRED_MEMBERS_ITEM. This type extends the USER_TYPE. It adds:

- MEMBERS
- REQUIRED_MEMBERS

If there are no members or required members an empty array is returned in the respective items.

124.12.1.5 public static final String MEMBERS = "Members"

The MEMBERS key, used in MEMBERS_ITEM.

124.12.1.6 public static final Item MEMBERS_ITEM

The item containing the members of a group. The key is MEMBERS and the type is 
JmxConstants.STRING_ARRAY_TYPE. It is used in GROUP_TYPE.

124.12.1.7 public static final String NAME = "Name"

The key NAME, used in NAME_ITEM.

124.12.1.8 public static final Item NAME_ITEM

The item for the user name for an authorization object. The key is NAME and the type is 
SimpleType.STRING.

124.12.1.9 public static final String OBJECTNAME = "osgi.compendium:service=useradmin,version=1.1"

User Admin MBean object name.

124.12.1.10 public static final String PROPERTIES = "Properties"

The PROPERTIES key, used in PROPERTIES_ITEM.
124.12.1.11  public static final Item PROPERTIES_ITEM
The item containing the properties of a Role. The key is PROPERTIES and the type is JmxConstants.PROPERTIES_TYPE.

124.12.1.12  public static final String REQUIRED_MEMBERS = "RequiredMembers"
The REQUIRED_MEMBERS key, used in REQUIRED_MEMBERS_ITEM.

124.12.1.13  public static final Item REQUIRED_MEMBERS_ITEM
The item containing the required members of a group. The key is REQUIRED_MEMBERS and the type is JmxConstants.STRING_ARRAY_TYPE. It is used in GROUP_TYPE.

124.12.1.14  public static final CompositeType ROLE_TYPE
The Composite Type for a Role. It contains the following items:

   • NAME
   • TYPE
   • PROPERTIES

124.12.1.15  public static final String ROLES = "Roles"
The key ROLES, used in ROLES_ITEM.

124.12.1.16  public static final Item ROLES_ITEM
The item containing the roles for this authorization object. The key is ROLES and the type is JmxConstants.STRING_ARRAY_TYPE.

124.12.1.17  public static final String TYPE = "Type"
The Role TYPE key, used in TYPE_ITEM.

124.12.1.18  public static final Item TYPE_ITEM
The item containing the type of the roles encapsulated by this authorization object. The key is TYPE and the type is SimpleType.INTEGER.

124.12.1.19  public static final CompositeType USER_TYPE
A Composite Type for a User. A User contains its Role description and adds the credentials. It extends ROLE_TYPE and adds CREDENTIALS_ITEM. This type extends the ROLE_TYPE. It adds:

   • CREDENTIALS

124.12.1.20  public void addCredential(String key,byte[] value,String username) throws IOException
   key The key of the credential to add
   value The value of the credential to add
   username The name of the user that gets the credential.
   □ Add credentials to a user, associated with the supplied key

   Throws IOException– if the operation fails
   IllegalArgumentException– if the username is not a User

124.12.1.21  public void addCredentialString(String key,String value,String username) throws IOException
   key The key of the credential to add
value  The value of the credential to add
username  The name of the user that gets the credential.
   □ Add credentials to a user, associated with the supplied key

Throws  IOException – if the operation fails
    IllegalArgumentException – if the username is not a User

124.12.1.22  public boolean addMember(String groupname, String rolename) throws IOException

groupname  The group name that receives the rolename as member.
rolename  The rolename (User or Group) that must be added.
   □ Add a member to the group.

Returns  true if the role was added to the group

Throws  IOException – if the operation fails
    IllegalArgumentException – if an invalid group name or role name is specified

124.12.1.23  public void addProperty(String key, byte[] value, String rolename) throws IOException

key  The added property key
value  The added byte[] property value
rolename  The role name that receives the property
   □ Add or update a property on a role.

Throws  IOException – if the operation fails
    IllegalArgumentException – if an invalid role name is specified

124.12.1.24  public void addPropertyString(String key, String value, String rolename) throws IOException

key  The key of the property to add
value  The value of the property to add (String)
rolename  The role name
   □ Add or update a property on a role

Throws  IOException – if the operation fails
    IllegalArgumentException – if an invalid role name is specified

124.12.1.25  public boolean addRequiredMember(String groupname, String rolename) throws IOException

groupname  The group name that is added
rolename  The role that
   □ Add a required member to the group

Returns  true if the role was added to the group

Throws  IOException – if the operation fails
    IllegalArgumentException – if an invalid group name or role name is specified

124.12.1.26  public void createGroup(String name) throws IOException

name  Name of the group to create
   □ Create a Group

Throws  IOException – if the operation fails
124.12.27  public void createRole(String name) throws IOException
    name Ignored.
    □ This method was specified in error and must not be used.
    Throws IOException—This method will throw an exception if called.
    Deprecated This method was specified in error. It does not function and must not be used. Use either createGroup(String) or createUser(String).

124.12.28  public void createUser(String name) throws IOException
    name Name of the user to create
    □ Create a User
    Throws IOException—if the operation fails

124.12.29  public CompositeData getAuthorization(String user) throws IOException
    user The user name
    □ Answer the authorization for the user name. The Composite Data is typed by AUTORIZATION_TYPE.
    Returns the Authorization typed by AUTORIZATION_TYPE.
    Throws IOException—if the operation fails
    IllegalArgumentException—if the user name is not a User

124.12.30  public TabularData getCredentials(String username) throws IOException
    username The user name
    □ Answer the credentials associated with a user. The returned Tabular Data is typed by JmxConstants.PROPERTIES_TYPE.
    Returns the credentials associated with the user, see JmxConstants.PROPERTIES_TYPE
    Throws IOException—if the operation fails
    IllegalArgumentException—if the user name is not a User

124.12.31  public CompositeData getGroup(String groupname) throws IOException
    groupname The group name
    □ Answer the Group associated with the group name. The returned Composite Data is typed by GROUP_TYPE
    Returns the Group, see GROUP_TYPE
    Throws IOException—if the operation fails
    IllegalArgumentException—if the group name is not a Group

124.12.32  public String[] getGroups(String filter) throws IOException
    filter The filter to apply
    □ Answer the list of group names
    Returns The list of group names
    Throws IOException—if the operation fails

124.12.33  public String[] getImpliedRoles(String username) throws IOException
    username The name of the user that has the implied roles
Answer the list of implied roles for a user

Returns: The list of role names

Throws:
- IOException – if the operation fails
- IllegalArgumentException – if the username is not a User

```java
124.12.1.34 public String[] getMembers(String groupname) throws IOException
groupname: The name of the group to get the members from

☐ Answer the the user names which are members of the group

Returns: The list of user names

Throws:
- IOException – if the operation fails
- IllegalArgumentException – if the groupname is not a Group
```

Answer the the user names which are members of the group

Returns: The list of user names

Throws:
- IOException – if the operation fails
- IllegalArgumentException – if the groupname is not a Group

```java
124.12.1.35 public TabularData getProperties(String rolename) throws IOException
rolename: The name of the role to get properties from

☐ Answer the properties associated with a role. The returned Tabular Data is typed by JmxConstants.PROPERTIES_TYPE.

Returns: the properties associated with the role, see JmxConstants.PROPERTIES_TYPE

Throws:
- IOException – if the operation fails
- IllegalArgumentException – if the rolename is not a role
```

Answer the properties associated with a role. The returned Tabular Data is typed by JmxConstants.PROPERTIES_TYPE.

Returns: the properties associated with the role, see JmxConstants.PROPERTIES_TYPE

Throws:
- IOException – if the operation fails
- IllegalArgumentException – if the rolename is not a role

```java
124.12.1.36 public String[] getRequiredMembers(String groupname) throws IOException
groupname: The name of the group to get the required members from

☐ Answer the list of user names which are required members of this group

Returns: The list of user names

Throws:
- IOException – if the operation fails
- IllegalArgumentException – if the group name is not a Group
```

Answer the list of user names which are required members of this group

Returns: The list of user names

Throws:
- IOException – if the operation fails
- IllegalArgumentException – if the group name is not a Group

```java
124.12.1.37 public CompositeData getRole(String name) throws IOException
name: The name of the role to get the data from

☐ Answer the role associated with a name. The returned Composite Data is typed by ROLE_TYPE.

Returns: the Role, see ROLE_TYPE

Throws:
- IOException – if the operation fails
- IllegalArgumentException – if the name is not a role
```

Answer the role associated with a name. The returned Composite Data is typed by ROLE_TYPE.

Returns: the Role, see ROLE_TYPE

Throws:
- IOException – if the operation fails
- IllegalArgumentException – if the name is not a role

```java
124.12.1.38 public String[] getRoles(String filter) throws IOException
filter: The string representation of the org.osgi.framework.Filter that is used to filter the roles by applying to the properties, if null all roles are returned.

☐ Answer the list of role names which match the supplied filter

Returns: The list the role names

Throws: IOException – if the operation fails
```

Answer the list of role names which match the supplied filter

Returns: The list the role names

Throws: IOException – if the operation fails

```java
124.12.1.39 public CompositeData getUser(String username) throws IOException
username: The name of the requested user

☐ Answer the list of implied roles for a user

Returns: The list of role names

Throws:
- IOException – if the operation fails
- IllegalArgumentException – if the username is not a User
```
Answer the User associated with the user name. The returned Composite Data is typed by USER_TYPE.

**Returns**
The User, see USER_TYPE

**Throws**
IllegalArgumentException— if the operation fails

124.12.1.40 public String[] getUsers(String filter) throws IOException

- **filter**
The filter to apply
- **Returns**
The list of user names
- **Throws**
IOException— if the operation fails

124.12.1.41 public String getUserWithProperty(String key,String value) throws IOException

- **key**
The key to compare
- **value**
The value to compare
- **Returns**
The User
- **Throws**
IOException— if the operation fails

124.12.1.42 public String[] listGroups() throws IOException

- **Returns**
The list of group names
- **Throws**
IOException— if the operation fails

124.12.1.43 public String[] listRoles() throws IOException

- **Returns**
The list of role names
- **Throws**
IOException— if the operation fails

124.12.1.44 public String[] listUsers() throws IOException

- **Returns**
The list of user names
- **Throws**
IOException— if the operation fails

124.12.1.45 public void removeCredential(String key,String username) throws IOException

- **key**
The key of the credential to remove
- **username**
The name of the user for which the credential must be removed
- **Returns**
Removes the credential associated with the given user
- **Throws**
IOException— if the operation fails

124.12.1.46 public boolean removeGroup(String name) throws IOException

- **name**

- **Returns**

- **Throws**
IOException— if the operation fails

IllegalArgumentException— if the username is not a User
Remove the Group associated with the name

**Returns**
true if the remove succeeded

**Throws**
IOException – if the operation fails
IllegalStateException – if the name is not a Group

public boolean removeMember(String groupname, String rolename) throws IOException

**groupname**
The group name

**rolename**

Remove a role from the group

**Returns**
true if the role was removed from the group

**Throws**
IOException – if the operation fails
IllegalStateException – if the groupname is not a Group

public void removeProperty(String key, String rolename) throws IOException

**key**

**rolename**

Remove a property from a role

**Throws**
IOException – if the operation fails
IllegalStateException – if the rolename is not a role

public boolean removeRole(String name) throws IOException

**name**

Remove the Role associated with the name

**Returns**
true if the remove succeeded

**Throws**
IOException – if the operation fails
IllegalStateException – if the name is not a role

public boolean removeUser(String name) throws IOException

**name**

Remove the User associated with the name

**Returns**
true if the remove succeeded

**Throws**
IOException – if the operation fails
IllegalStateException – if the name is not a User

**124.13**

**org.osgi.jmx.framework.wiring**


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.jmx.framework.wiring; version="[1.1,2.0)"
Example import for providers implementing the API in this package:

```java
import-package: org.osgi.jmx.framework.wiring; version="[1.1,1.2)"
```

### 124.13.1 public interface BundleWiringStateMBean

This MBean represents the bundle wiring state. It can be used to retrieve the declared capabilities, declared requirements, and wiring for the current and past revisions of bundles.

**Concurrency** Thread-safe

#### 124.13.1.1 public static final String ATTRIBUTES = "Attributes"
The key of ATTRIBUTES_ITEM.

#### 124.13.1.2 public static final Item ATTRIBUTES_ITEM
The item containing the attributes of a capability or requirement. Used in BUNDLE_REQUIREMENT_TYPE and BUNDLE_CAPABILITY_TYPE. The key is ATTRIBUTES and the type is ATTRIBUTES_TYPE.

#### 124.13.1.3 public static final TabularType ATTRIBUTES_TYPE
The Tabular Type that holds the attributes for a capability or requirements. The row type is JmxConstants.PROPERTY_TYPE and the index is JmxConstants.KEY.

#### 124.13.1.4 public static final String BUNDLE_CAPABILITY = "BundleCapability"
The key of BUNDLE_CAPABILITY_ITEM.

#### 124.13.1.5 public static final Item BUNDLE_CAPABILITY_ITEM
The item containing a capability for a bundle in BUNDLE_WIRE_TYPE. The key is BUNDLE_CAPABILITY and the type is BUNDLE_CAPABILITY_TYPE.

#### 124.13.1.6 public static final CompositeType BUNDLE_CAPABILITY_TYPE
The Composite Type that represents the capability of a bundle. The composite consists of:
- NAMESPACE
- ATTRIBUTES
- DIRECTIVES

#### 124.13.1.7 public static final String BUNDLE_ID = "BundleId"
The key of BUNDLE_ID_ITEM.

#### 124.13.1.8 public static final Item BUNDLE_ID_ITEM
The item containing a bundle ID. They key is BUNDLE_ID and the type is a long.

#### 124.13.1.9 public static final String BUNDLE_REQUIREMENT = "BundleRequirement"
The key of BUNDLE_REQUIREMENT_ITEM.

#### 124.13.1.10 public static final Item BUNDLE_REQUIREMENT_ITEM
The item containing a requirement for a bundle in BUNDLE_WIRE_TYPE. The key is BUNDLE_REQUIREMENT and the type is BUNDLE_REQUIREMENT_TYPE.

#### 124.13.1.11 public static final CompositeType BUNDLE_REQUIREMENT_TYPE
The Composite Type that represents the requirement of a bundle. The composite consists of:
- NAMESPACE
public static final String BUNDLE_REVISION_ID = "BundleRevisionId"

The key of BUNDLE_REVISION_ID_ITEM.

public static final Item BUNDLE_REVISION_ID_ITEM

The item containing a bundle revision ID. A bundle revision ID is always local to the result of a JMX invocation and do not have a defined meaning across invocation calls. They are used where a result can potentially contain multiple revisions of the same bundle. The key is BUNDLE_REVISION_ID and the type is an integer.

public static final CompositeType BUNDLE_WIRE_TYPE

The Composite type that represents a bundle wire describing the live association between a provider of a capability and a requirer of the corresponding requirement.

The composite consists of:

- BUNDLE_REQUIREMENT
- BUNDLE_CAPABILITY
- PROVIDER_BUNDLE_ID
- PROVIDER_BUNDLE_REVISION_ID
- REQUIRER_BUNDLE_ID
- REQUIRER_BUNDLE_REVISION_ID

public static final ArrayType BUNDLE_WIRES_TYPE_ARRAY

An array of BUNDLE_WIRE_TYPES.

public static final CompositeType BUNDLE_WIRING_TYPE

The Composite Type that represents a bundle wiring. The composite consists of:

- BUNDLE_ID
- BUNDLE_REVISION_ID
- REQUIREMENTS
- CAPABILITIES
- REQUIRED_WIRES
- PROVIDED_WIRES

public static final TabularType BUNDLES_WIRING_TYPE

The Tabular Type to hold the wiring of a number of bundles. The row type is BUNDLE_WIRING_TYPE and the index is the combination of the BUNDLE_ID and the BUNDLE_REVISION_ID.

public static final String CAPABILITIES = "Capabilities"

The key of CAPABILITIES_ITEM.

public static final Item CAPABILITIES_ITEM

The item containing the capabilities in REVISION_CAPABILITIES_TYPE and BUNDLE_WIRING_TYPE. The key is CAPABILITIES and the type is CAPABILITY_TYPE_ARRAY.

public static final ArrayType CAPABILITY_TYPE_ARRAY

An array of BUNDLE_CAPABILITY_TYPES.
124.13.1.21  public static final CompositeType DIRECTIVE_TYPE
The Composite Type that represents a directive for a capability or requirement. The composite consists of:
  • KEY
  • VALUE

124.13.1.22  public static final String DIRECTIVES = "Directives"
The key of DIRECTIVES_ITEM.

124.13.1.23  public static final Item DIRECTIVES_ITEM
The item containing the directives of a capability or requirement. Used in BUNDLE_REQUIREMENT_TYPE and BUNDLE_CAPABILITY_TYPE. The key is DIRECTIVES and the type is DIRECTIVES_TYPE.

124.13.1.24  public static final TabularType DIRECTIVES_TYPE
The Tabular Type that holds the directives for a capability or requirement. The row type is DIRECTIVE_TYPE and the index is KEY.

124.13.1.25  public static final String KEY = "Key"
The key of KEY_ITEM.

124.13.1.26  public static final Item KEY_ITEM
The item containing the key of a capability or requirement directive. Used in DIRECTIVE_TYPE. The key is KEY and the type is a String.

124.13.1.27  public static final String NAMESPACE = "Namespace"
The key of NAMESPACE_ITEM.

124.13.1.28  public static final Item NAMESPACE_ITEM
The item containing the namespace for a capability or requirement. Used in BUNDLE_REQUIREMENT_TYPE and BUNDLE_CAPABILITY_TYPE. The key is NAMESPACE and the type is a String.

124.13.1.29  public static final String OBJECTNAME = "osgi.core:type=wiringState,version=1.1"
The Object Name prefix for this mbean. The full object name also contains the framework name and uuid as properties.

124.13.1.30  public static final String PROVIDED_WIRES = "ProvidedWires"
The key of PROVIDED_WIRES_ITEM.

124.13.1.31  public static final Item PROVIDED_WIRES_ITEM
The item containing the provided wires in BUNDLE_WIRING_TYPE. The key is PROVIDED_WIRES and the type is BUNDLE_WIRES_TYPE_ARRAY.

124.13.1.32  public static final String PROVIDER_BUNDLE_ID = "ProviderBundleId"
The key of PROVIDER_BUNDLE_ID_ITEM.

124.13.1.33  public static final Item PROVIDER_BUNDLE_ID_ITEM
The item containing the provider bundle ID in BUNDLE_WIRE_TYPE. The key is PROVIDER_BUNDLE_ID and the type is a long.
124.13.1.34 public static final String PROVIDER_BUNDLE_REVISION_ID = "ProviderBundleRevisionId"
The key of PROVIDER_BUNDLE_REVISION_ID_ITEM.

124.13.1.35 public static final Item PROVIDER_BUNDLE_REVISION_ID_ITEM
The local ID of a provider revision in BUNDLE_WIRE_TYPE. This ID is local to the result where it resides and has no defined meaning across multiple invocations. The key is PROVIDER_BUNDLE_REVISION_ID and the type is an int.

124.13.1.36 public static final String REQUIRED_WIRES = "RequiredWires"
The key of REQUIRED_WIRES_ITEM.

124.13.1.37 public static final Item REQUIRED_WIRES_ITEM
The item containing the required wires in BUNDLE_WIRING_TYPE. The key is REQUIRED_WIRES and the type is BUNDLE_WIRES_TYPE_ARRAY.

124.13.1.38 public static final ArrayType REQUIREMENT_TYPE_ARRAY
An array of BUNDLE_REQUIREMENT_TYPEs.

124.13.1.39 public static final String REQUIREMENTS = "Requirements"
The key of REQUIREMENTS_ITEM.

124.13.1.40 public static final Item REQUIREMENTS_ITEM
The item containing the requirements in REVISION_REQUIREMENTS_TYPE and BUNDLE_WIRING_TYPE. The key is REQUIREMENTS and the type is REQUIREMENT_TYPE_ARRAY.

124.13.1.41 public static final String REQUIRER_BUNDLE_ID = "RequirerBundleId"
The key of REQUIRER_BUNDLE_ID_ITEM.

124.13.1.42 public static final Item REQUIRER_BUNDLE_ID_ITEM
The item containing the requirer bundle ID in BUNDLE_WIRE_TYPE. The key is REQUIRER_BUNDLE_ID and the type is long.

124.13.1.43 public static final String REQUIRER_BUNDLE_REVISION_ID = "RequirerBundleRevisionId"
The key of REQUIRER_BUNDLE_REVISION_ID_ITEM.

124.13.1.44 public static final Item REQUIRER_BUNDLE_REVISION_ID_ITEM
The local ID of a requirer revision in BUNDLE_WIRE_TYPE. This ID is local to the result where it resides and has no defined meaning across multiple invocations. The key is REQUIRER_BUNDLE_REVISION_ID and the type is an int.

124.13.1.45 public static final CompositeType REVISION_CAPABILITIES_TYPE
The Composite Type that represents the capabilities for a revision. The composite consists of:
- BUNDLE_REVISION_ID
- CAPABILITIES

124.13.1.46 public static final CompositeType REVISION_REQUIREMENTS_TYPE
The Composite Type that represents the requirements of a revision. The composite consists of:
- BUNDLE_REVISION_ID
• REQUIREMENTS

124.13.1.47 public static final TabularType REVISIONS_CAPABILITIES_TYPE
The Tabular Type that holds the capabilities of a revision. The row type is REVISION_CAPABILITIES_TYPE and the index is BUNDLE_REVISION_ID.

124.13.1.48 public static final TabularType REVISIONS_REQUIREMENTS_TYPE
The Tabular Type that hold the requirements of a revision. The row type is REVISION_REQUIREMENTS_TYPE and the index is BUNDLE_REVISION_ID.

124.13.1.49 public static final String VALUE = "Value"
The key of VALUE.

124.13.1.50 public static final Item VALUE_ITEM
The item containing the value of a capability or requirement directive. Used in DIRECTIVE_TYPE. They key is VALUE and the type is a String.

124.13.1.51 public CompositeData[] getCurrentRevisionDeclaredCapabilities(long bundleId, String namespace) throws IOException, JMException

bundleId The bundle ID.
namespace The namespace of the capabilities to be returned by this operation.

 Returns the declared capabilities for the current revision of bundleId and namespace.
Throws JMException – if there is a JMX problem.
IOException – if the connection could not be made because of a communication problem.
See Also for the details of the CompositeData.

124.13.1.52 public CompositeData[] getCurrentRevisionDeclaredRequirements(long bundleId, String namespace) throws IOException, JMException

bundleId The bundle ID.
namespace The namespace of the requirements to be returned by this operation.

 Returns the declared requirements for the current revision of bundleId and namespace.
Throws JMException – if there is a JMX problem.
IOException – if the connection could not be made because of a communication problem.
See Also for the details of the CompositeData.

124.13.1.53 public CompositeData getCurrentWiring(long bundleId, String namespace) throws IOException, JMException

bundleId The bundle ID.
namespace The namespace of the requirements and capabilities for which to return information.

 Returns the bundle wiring for the current bundle revision.
Throws JMException – if there is a JMX problem.
IOException – if the connection could not be made because of a communication problem.
124.13.1.54 public TabularData getCurrentWiringClosure(long rootBundleId, String namespace) throws IOException, JMException

rootBundleId: the root bundle of the closure.
namespace: The namespace of the requirements and capabilities for which to return information.

Returns: a tabular representation of all the wirings in the closure. The bundle revision IDs only have meaning in the context of the current result. The revision of the root bundle is set to 0. Therefore the root bundle of the closure can be looked up in the table by its bundle ID and revision 0.

Throws: JMException – if there is a JMX problem.
IOException – if the connection could not be made because of a communication problem.

See Also: for the details of TabularData.

124.13.1.55 public TabularData getRevisionsDeclaredCapabilities(long bundleId, String namespace) throws IOException, JMException

bundleId: The bundle ID.
namespace: The namespace of the capabilities to be returned by this operation.

Returns: the declared capabilities for all revisions of bundleId.

Throws: JMException – if there is a JMX problem.
IOException – if the connection could not be made because of a communication problem.

See Also: for the details of TabularData. The capabilities are in no particular order, and may change in subsequent calls to this operation.

124.13.1.56 public TabularData getRevisionsDeclaredRequirements(long bundleId, String namespace) throws IOException, JMException

bundleId: The bundle ID.
namespace: The namespace of the requirements to be returned by this operation.

Returns: the declared requirements for all revisions of bundleId.

Throws: JMException – if there is a JMX problem.
IOException – if the connection could not be made because of a communication problem.

See Also: for the details of TabularData. The requirements are in no particular order, and may change in subsequent calls to this operation.

124.13.1.57 public TabularData getRevisionsWiring(long bundleId, String namespace) throws IOException, JMException

bundleId: The bundle ID.
namespace: The namespace of the requirements and capabilities for which to return information.

Returns: the wiring information for all revisions of bundleId and namespace.

Throws: JMException – if there is a JMX problem.
IOException – if the connection could not be made because of a communication problem.

See Also for the details of TabularData. The bundle wirings are in no particular order, and may change in subsequent calls to this operation.

124.13.1.58 public TabularData getRevisionsWiringClosure(long rootBundleId, String namespace) throws IOException, JMException

rootBundleId The root bundle ID.

namespace The namespace of the requirements and capabilities for which to return information.

Returns the bundle wiring closure for all revisions of the specified bundle. The wiring closure contains all the wirings from the root bundle revision to all bundle revisions it is wired to and all their transitive wirings.

Returns a tabular representation of all the wirings in the closure. The bundle revision IDs only have meaning in the context of the current result.

Throws JMException – if there is a JMX problem.
IOException – if the connection could not be made because of a communication problem.

See Also for the details of TabularData. The bundle wirings are in no particular order, and may change in subsequent calls to this operation. Furthermore, the bundle revision IDs are local and cannot be reused across invocations.

124.14 References

[1] JMX
http://en.wikipedia.org/wiki/JMX

http://docs.oracle.com/javase/1.5.0/docs/guide/jmx/overview/JMXoverviewTOC.html

http://www.jcp.org/en/jsr/detailid=3

http://www.jcp.org/en/jsr/detailid=255


http://www.jcp.org/en/jsr/detailid=262

http://docs.oracle.com/javase/1.5.0/docs/guide/jmx/spec.html

[8] Using JConsole to Monitor Applications
JDBC™ Service Specification

Version 1.0

125.1 Introduction

The Java Database Connectivity (JDBC) standard provides an API for applications to interact with relational database systems from different vendors. To abstract over concrete database systems and vendor specific characteristics, the JDBC specification provides various classes and Service Provider Interfaces (SPI) that can be used for database interaction. Implementations are database specific and provided by the corresponding driver. This specification defines how OSGi-aware JDBC drivers can provide access to their implementations. Applications can rely on this mechanism to transparently access drivers and to stay independent from driver specific classes. Additionally, this mechanism helps to use common OSGi practices and to avoid class loading problems.

This specification uses a number of packages that are defined in Java SE 1.4 or later.

125.1.1 Essentials

- **Registration** - Provide a mechanism for JDBC driver announcements.
- **Lookup** - Inspect available database drivers and provide means for driver access.
- **Services** - Uses a service model for getting the driver objects.
- **Compatible** - Minimize the amount of work needed to support this specification for existing drivers.

125.1.2 Entities

- **Relational Database Management Systems** (RDBMS) - An external database system.
- **Database Driver** - JDBC-compliant database driver that is delivered in a bundle.
- **Data Source Factory** - Provides one of the different Data Sources that gives access to a database driver.
- **Application** - The application that wants to access a relational database system.

Figure 125.1  JDBC Class/Service Overview
125.1.3 Dependencies

The classes and interfaces used in this specification come from the following packages:

javax.sql
java.sql

These packages have no associated version. It is assumed they come from the runtime environment. This specification is based on Java SE 1.4 or later.

125.1.4 Synopsis

A JDBC Database Driver is the software that maps the JDBC specification to a specific implementation of a relational database. For OSGi, JDBC drivers are delivered as driver bundles. A driver bundle registers a Data Source Factory service when it is ACTIVE. Service properties are used to specify the database driver name, version, etc. The Data Source Factory service provides methods to create DataSource, ConnectionPoolDataSource, XADataSource, or Driver objects. These objects are then used by an application to interact with the relational database system in the standard way.

The application can query the service registry for available Data Source Factory services. It can select particular drivers by filtering on the service properties. This service based model is easy to use with dependency injection frameworks like Blueprint or Declarative Services.

125.2 Database Driver

A Database Driver provides the connection between an Application and a particular database. A single OSGi Framework can contain several Database Drivers simultaneously. To make itself available to Applications, a Database Driver must register a Data Source Factory service. Applications must be able to find the appropriate Database Driver. The Database Driver must therefore register the Data Source Factory service with the following service properties:

- **OSGI_JDBC_DRIVER_CLASS** - (String) The required name of the driver implementation class. This property is the primary key to find a driver's Data Source Factory. It is not required that there is an actual class with this name.
- **OSGI_JDBC_DRIVER_NAME** - (String) The optional driver name. This property is informational.
- **OSGI_JDBC_DRIVER_VERSION** - (String) The driver version. The version is not required to be an OSGi version, it should be treated as an opaque string. This version is likely not related to the package of the implementation class or its bundle.

The previous properties are vendor-specific and are meant to further describe the Database Driver to the Application.

Each Data Source Factory service must relate to a single Database Driver. The Database Driver implementation bundle does not necessarily need to be the registrar of the Data Source Factory service. Any bundle can provide the Data Source Factory service and delegate to the appropriate driver specific implementation classes. However, as JDBC driver implementations evolve to include built-in support for OSGi they can provide the Data Source Factory service themselves. This implies that the same driver can be registered multiple times.

125.2.1 Life Cycle

A Data Source Factory service should be registered while its Driver Bundle is in the ACTIVE state or when it has a lazy activation policy and is in the STARTING state.

What happens to the objects created by the Data Source Factory service, and the objects they created, is undefined in this specification. Database Drivers are not mandated to track the proper life cycle of these objects.
125.2.2 Package Dependencies

A Database Driver must import the javax.sql package. The java.sql package that contains the Driver and SQLException interface is automatically imported because it starts with java.. Both packages are contained in the JRE since Java SE 1.4. These packages are not normally versioned with OSGi version numbers. Bundles using the Data Source Factory must therefore ensure they get the proper imports, which is usually from the JRE. Due to the lack of specified metadata, the deployer is responsible for ensuring this.

125.3 Applications

125.3.1 Selecting the Data Source Factory Service

Applications can query the OSGi service registry for available Database Drivers by getting a list of Data Source Factory services. Normally, the application needs access to specific drivers that match their needed relational database type. The service properties can be used to find the desired Database Driver. This model is well supported by dependency injection frameworks like Blueprint or Declarative Services. However, it can of course also be used with the basic service methods. The following code shows how a Service Tracker can be used to get a Database Driver called ACME DB.

```java
Filter filter = context.createFilter(
   "(&(objectClass=" +
   DataSourceFactory.class.getName() +
   ")(" +
   DataSourceFactory.OSGI_JDBC_DRIVER_CLASS + "=com.acme.db.Driver))");

ServiceTracker tracker = new ServiceTracker(context, filter, null);
tracker.open();

DataSourceFactory dsf = (DataSourceFactory) tracker.getService();
```

125.3.2 Using Database Drivers

The Data Source Factory service can be used to obtain instances for the following JDBC related types:

- javax.sql.DataSource
- javax.sql.ConnectionPoolDataSource
- javax.sql.XADataSource
- java.sql.Driver

Which type of Connection provider that is actually required depends on the Application and the use case. For each type, the Data Source Factory service provides a method that returns the corresponding instance. Each method takes a Properties object as a parameter to pass a configuration to the Database Driver implementation. The configuration is driver-specific and can be used to specify the URL for the database and user credentials. Common property names for these configuration properties are also defined in the DataSourceFactory interface.

A Data Source Factory is not required to implement all of the factory methods. If an implementation does not support a particular type then it must throw a SQL Exception. This specification does not provide a mechanism to depend on a Data Source Factory service that implements a particular factory method.

The following code shows how a DataSource object could be created.

```java
Properties props = new Properties();
```
props.put(DataSourceFactory.JDBC_URL, "jdbc:acme:ACMEDB");
props.put(DataSourceFactory.JDBC_USER, "foo");
props.put(DataSourceFactory.JDBC_PASSWORD, "secret");
DataSource dataSource = dsf.createDataSource(props);

The DataSourceFactory interface has several static fields that represent common property keys for the Properties instance. General properties are:

- JDBC_DATABASE_NAME
- JDBC_DATASOURCE_NAME
- JDBC_DESCRIPTION
- JDBC_NETWORK_PROTOCOL
- JDBC_PASSWORD
- JDBC_PORT_NUMBER
- JDBC_ROLE_NAME
- JDBC_SERVER_NAME
- JDBC_USER
- JDBC_URL

The following additional property keys are provided for applications that want to create a ConnectionPoolDataSource object or a XAPoolDataSource object:

- JDBC_INITIAL_POOL_SIZE
- JDBC_MAX_IDLE_TIME
- JDBC_MAX_POOL_SIZE
- JDBC_MAX_STATEMENTS
- JDBC_MIN_POOL_SIZE
- JDBC_PROPERTY_CYCLE

Which property keys and values are supported depends on the driver implementation. Drivers can support additional custom configuration properties.

125.3.3 Using JDBC in OSGi and Containers

The JDBC service provides JDBC driver services, not container services. A typical client would only use the DataSourceFactory.createDataSource() method to procure a regular Data Source from which they can obtain (usually non-pooled) connections.

Containers generally offer connection pools and support XA transactions. The container manages the pools and does this by using Pooled Connection or XA Connection objects from a driver-implemented respective Connection Pool Data Source or XA Data Source. To support containers, frameworks, or any client that wants to manage a pool, these Data Source types are included in the DataSource Factory service. Drivers are permitted to implement their own Data Source using an underlying connection pooling scheme. This is driver-dependent and not related to the OSGi specifications.

The usual set of JDBC properties are defined in the services for use with the Data Source types. They are the same as what is defined for JDBC and the caller should know which properties make sense when passed to a given Data Source type. The same result should occur in OSGi as occurs outside of OSGi. If the driver does not support a given property with a given Data Source type then it can ignore it or it can throw an Exception.

125.4 Security

This specification depends on the JDBC specification for security.
125.5 org.osgi.service.jdbc

JDBC 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. 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.jdbc; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:
```
Import-Package: org.osgi.service.jdbc; version="[1.0,1.1)"
```

125.5.1 public interface DataSourceFactory

A factory for JDBC connection factories. There are 3 preferred connection factories for getting JDBC connections: `javax.sql.DataSource`, `javax.sql.ConnectionPoolDataSource`, and `javax.sql.XADataSource`. DataSource providers should implement this interface and register it as an OSGi service with the JDBC driver class name in the OSGI_JDBC_DRIVER_CLASS property.

Concurrency  Thread-safe

125.5.1.1 public static final String JDBC_DATABASE_NAME = "databaseName"

The "databaseName" property that DataSource clients should supply a value for when calling `createDataSource(Properties)`.

125.5.1.2 public static final String JDBC_DATASOURCE_NAME = "dataSourceName"

The "dataSourceName" property that DataSource clients should supply a value for when calling `createDataSource(Properties)`.

125.5.1.3 public static final String JDBC_DESCRIPTION = "description"

The "description" property that DataSource clients should supply a value for when calling `createDataSource(Properties)`.

125.5.1.4 public static final String JDBC_INITIAL_POOL_SIZE = "initialPoolSize"

The "initialPoolSize" property that ConnectionPoolDataSource and XADataSource clients may supply a value for when calling `createConnectionPoolDataSource(Properties)` or `createXADataSource(Properties)` on drivers that support this property.

125.5.1.5 public static final String JDBC_MAX_IDLE_TIME = "maxIdleTime"

The "maxIdleTime" property that ConnectionPoolDataSource and XADataSource clients may supply a value for when calling `createConnectionPoolDataSource(Properties)` or `createXADataSource(Properties)` on drivers that support this property.

125.5.1.6 public static final String JDBC_MAX_POOL_SIZE = "maxPoolSize"

The "maxPoolSize" property that ConnectionPoolDataSource and XADataSource clients may supply a value for when calling `createConnectionPoolDataSource(Properties)` or `createXADataSource(Properties)` on drivers that support this property.

125.5.1.7 public static final String JDBC_MAX_STATEMENTS = "maxStatements"

The "maxStatements" property that ConnectionPoolDataSource and XADataSource clients may supply a value for when calling `createConnectionPoolDataSource(Properties)` or `createXADataSource(Properties)` on drivers that support this property.
125.5.1.8 public static final String JDBC_MIN_POOL_SIZE = "minPoolSize"

The "minPoolSize" property that ConnectionPoolDataSource and XADataSource clients may supply a value for when calling createConnectionPoolDataSource(Properties) or createXADataSource(Properties) on drivers that support this property.

125.5.1.9 public static final String JDBC_NETWORK_PROTOCOL = "networkProtocol"

The "networkProtocol" property that DataSource clients should supply a value for when calling createDataSource(Properties).

125.5.1.10 public static final String JDBC_PASSWORD = "password"

The "password" property that DataSource clients should supply a value for when calling createDataSource(Properties).

125.5.1.11 public static final String JDBC_PORT_NUMBER = "portNumber"

The "portNumber" property that DataSource clients should supply a value for when calling createDataSource(Properties).

125.5.1.12 public static final String JDBC_PROPERTY_CYCLE = "propertyCycle"

The "propertyCycle" property that ConnectionPoolDataSource and XADataSource clients may supply a value for when calling createConnectionPoolDataSource(Properties) or createXADataSource(Properties) on drivers that support this property.

125.5.1.13 public static final String JDBC_ROLE_NAME = "roleName"

The "roleName" property that DataSource clients should supply a value for when calling createDataSource(Properties).

125.5.1.14 public static final String JDBC_SERVER_NAME = "serverName"

The "serverName" property that DataSource clients should supply a value for when calling createDataSource(Properties).

125.5.1.15 public static final String JDBC_URL = "url"

The "url" property that DataSource clients should supply a value for when calling createDataSource(Properties).

125.5.1.16 public static final String JDBC_USER = "user"

The "user" property that DataSource clients should supply a value for when calling createDataSource(Properties).

125.5.1.17 public static final String OSGI_JDBC_DRIVER_CLASS = "osgi.jdbc.driver.class"

Service property used by a JDBC driver to declare the driver class when registering a JDBC DataSourceFactory service. Clients may filter or test this property to determine if the driver is suitable, or the desired one.

125.5.1.18 public static final String OSGI_JDBC_DRIVER_NAME = "osgi.jdbc.driver.name"

Service property used by a JDBC driver to declare the driver name when registering a JDBC DataSourceFactory service. Clients may filter or test this property to determine if the driver is suitable, or the desired one.

125.5.1.19 public static final String OSGI_JDBC_DRIVER_VERSION = "osgi.jdbc.driver.version"

Service property used by a JDBC driver to declare the driver version when registering a JDBC DataSourceFactory service. Clients may filter or test this property to determine if the driver is suitable, or the desired one.
125.5.1.20 public ConnectionPoolDataSource createConnectionPoolDataSource(Properties props) throws SQLException

props The properties used to configure the ConnectionPoolDataSource. null indicates no properties. If the property cannot be set on the ConnectionPoolDataSource being created then a SQLException must be thrown.

□ Create a new ConnectionPoolDataSource using the given properties.

Returns A configured ConnectionPoolDataSource.

Throws SQLException – If the ConnectionPoolDataSource cannot be created.

125.5.1.21 public DataSource createDataSource(Properties props) throws SQLException

props The properties used to configure the DataSource. null indicates no properties. If the property cannot be set on the DataSource being created then a SQLException must be thrown.

□ Create a new DataSource using the given properties.

Returns A configured DataSource.

Throws SQLException – If the DataSource cannot be created.

125.5.1.22 public Driver createDriver(Properties props) throws SQLException

props The properties used to configure the Driver. null indicates no properties. If the property cannot be set on the Driver being created then a SQLException must be thrown.

□ Create a new Driver using the given properties.

Returns A configured Driver.

Throws SQLException – If the Driver cannot be created.

125.5.1.23 public XADataSource createXADataSource(Properties props) throws SQLException

props The properties used to configure the XADataSource. null indicates no properties. If the property cannot be set on the XADataSource being created then a SQLException must be thrown.

□ Create a new XADataSource using the given properties.

Returns A configured XADataSource.

Throws SQLException – If the XADataSource cannot be created.

125.6 References

[1] Java SE 1.4
http://www.oracle.com/technetwork/java/archive-139210.html
126 JNDI Services Specification

Version 1.0

126.1 Introduction

Naming and directory services have long been useful tools in the building of software systems. The ability to use a programming interface to publish and consume objects can provide many benefits to any system. The Java Naming and Directory Interface (JNDI) is a registry technology in Java applications, both in the Java SE and Java EE space. JNDI provides a vendor-neutral set of APIs that allow clients to interact with a naming service from different vendors.

The JNDI as used in the Java SE environment relies on the class loading model provided by the JDK to find providers. By default, it attempts to load the JNDI provider class using the Thread Context Class Loader. In an OSGi environment, this type of Context creation is not desirable since it relies on the JNDI provider classes being visible to the JNDI client, or require it to set the Context Class Loader; in both cases breaking modularity. For modularity reasons, it is important that clients are not required to express a dependency on the implementation of services they use.

This specification will define how JNDI can be utilized from within an OSGi framework. The specification consists of three key parts:

- **OSGi Service Model** - How clients interact with JNDI when running inside an OSGi Framework.
- **JNDI Provider Model** - How JNDI providers can advertise their existence so they are available to OSGi and traditional clients.
- **Traditional Model** - How traditional JNDI applications and providers can continue to work in an OSGi Framework without needing to be rewritten when certain precautions are taken.

126.1.1 Essentials

- **Naming Service** - Provide an integration model for JNDI API clients and providers.
- **Flexible** - Provide a standard mechanism for publishing and locating JNDI providers.
- **Compatibility** - Support the traditional JNDI programming model used by Java SE and Java EE clients.
- **Service Based** - Provide a service model that clients and providers can use to leverage JNDI facilities.
- **Migration** - Provide a mechanism to access OSGi services from a JNDI context.

126.1.2 Entities

- **JNDI Implementation** - The Implementer of the JNDI Context Manager, JNDI Provider Admin, and setter of the JNDI static singletons.
- **JNDI Client** - Any code running within an OSGi bundle that needs to use JNDI.
- **JNDI Context Manager** - A service that allows clients to obtain Contexts via a service.
- **JNDI Provider Admin** - A service that allows the conversion of objects for providers.
- **JNDI Provider** - Provides a Context implementation.
- **Context** - A Context abstracts a namespace. Implementations are provided by JNDI providers and the Contexts are used by JNDI clients. The corresponding interface is javax.naming.Context.
• **Dir Context** - A sub-type of Context that provides mechanisms for examining and updating the attributes of an object in a directory structure, and for performing searches in a hierarchical naming systems like LDAP. The corresponding interface is `javax.naming.directory.DirContext`.

• **Initial Context Factory** - A factory for creating instances of Context objects. This factory is used to integrate new JNDI Providers. In general, a single Initial Context Factory constructs Context objects for a single provider implementation. The corresponding interface is `javax.naming.spi.InitialContextFactory`.

• **Initial Context Factory Builder** - A factory for InitialContextFactory objects. A single Initial Context Factory Builder can construct InitialContextFactory objects for different types of Contexts. The interface is `javax.naming.spi.InitialContextFactoryBuilder`.

• **Object Factory** - Used in conversion of objects. The corresponding interface is `javax.naming.spi.ObjectFactory`.

• **Dir Object Factory** - An Object Factory that takes attribute information for object conversion. The corresponding interface is `javax.naming.spi.DirObjectFactory`.

• **Object Factory Builder** - A factory for ObjectFactory objects. A single Object Factory Builder can construct ObjectFactory instances for different types of conversions. The corresponding interface is `javax.naming.spi.ObjectFactoryBuilder`.

• **Reference** - A description of an object that can be turned into an object through an Object Factory. The associated `Referenceable` interface implemented on an object indicates that it can provide a Reference object.

---

**Figure 126.1 JNDI Service Specification Service Entities**

---

### 126.1.3 Dependencies

The classes and interfaces used in this specification come from the following packages:

```java
javax.naming
javax.naming.spi
javax.naming.directory
```
126.1.4 Synopsis

A client bundle wishing to make use of JNDI in order to access JNDI Providers such as LDAP or DNS in OSGi should not use the Naming Manager but instead use the JNDI Context Manager service. This service can be asked for a Context based on environment properties. The environment properties are based on an optional argument in the newInitialContext method, the Java System properties, and an optional resource in the caller's bundle.

These environment properties can specify an implementation class name for a factory that can create a Context object. If such a class name is specified, then it is searched for in the service registry. If such a service is found, then that service is used to create a new Context, which is subsequently returned. If no class name is specified, the service registry is searched for Initial Context Factory services. These services are tried in ranking order to see if they can create an appropriate Context, the first one that can create a Context is then used.

If no class name is specified, all Initial Context Factory Builder services are tried to see if they can create a Context, the first non-null result is used. If no context can be found, a No Initial Context Exception is thrown. Otherwise, the JNDI Context Manager service returns an initial Context that uses the just created Context from a provider as the backing service. This initial Context delegates all operations to this backing Context, except operations that use a name that can be interpreted as a URL, that is, the name contains a colon. URL operations are delegated a URL Context that is associated with the used scheme. URL Contexts are found through the general object conversion facility provided by the JNDI Provider Admin service.

The JNDI Provider Admin service provides a general object conversion facility that can be extended with Object Factory and Object Factory Builder services that are traditionally provided through the Naming Manager getObjectInstance method. A specific case for this conversion is the use of Reference objects. Reference objects can be used to store objects persistently in a Context implementation. Reference objects must be converted to their corresponding object when retrieved from a Context.

During the client's use of a Context it is possible that its provider's service is unregistered. In this case the JNDI Context Manager must release the backing Context. If the initial Context is used and no backing Context is available, the JNDI Context Manager must re-create a new Context, if possible. Otherwise a Naming Exception is thrown. If subsequently a proper new backing Context can be created, the initial Context must start operating again.

The JNDI Context Manager service must track the life cycle of a calling bundle and ensure that any returned Context objects are closed and returned objects are properly cleaned up when the bundle is closed or the JNDI Context Manager service is unget.

When the client bundle is stopped, any returned initial Context objects are closed and discarded. If the Initial Context Factory, or Initial Context Factory Builder, service that created the initial Context goes away then the JNDI Context Manager service releases the Context backing the initial Context and attempts to create a replacement Context.

Clients and JNDI Context providers that are unaware of OSGi use static methods to connect to the JRE JNDI implementation. The InitialContext class provides access to a Context from a provider and providers use the static NamingManager methods to do object conversion and find URL Contexts. This traditional model is not aware of OSGi and can therefore only be used reliably if the consequences of this lack of OSGi awareness are managed.
126.2 JNDI Overview

The Java Naming and Directory Interface (JNDI) provides an abstraction for namespaces that is included in Java SE. This section describes the basic concepts of JNDI as provided in Java SE. These concepts are later used in the service model provided by this specification.

126.2.1 Context and Dir Context

The [1] Java Naming and Directory Interface (JNDI) defines an API for namespaces. These namespaces are abstracted with the Context interface. Namespaces that support attributes, such as a namespace as the Lightweight Directory Access Protocol (LDAP), are represented by the DirContext class, which extends the Context class. If applicable, a Context object can be cast to a DirContext object. The distinction is not relevant for this specification, except in places where it is especially mentioned.

The Context interface models a set of name-to-object bindings within a namespace. These bindings can be looked-up, created, and updated through the Context interface. The Context interface can be used for federated, flat, or hierarchical namespaces.

126.2.2 Initial Context

Obtaining a Context for a specific namespace, for example DNS, is handled through the InitialContext class. Creating an instance of this class will cause the JRE to find a backing Context. The InitialContext is only a facade for the backing Context. The facade context provides URL based lookups.

The backing Context is created by a JNDI Provider. How this backing Context is created is an elaborate process using class loading techniques or a provisioning mechanism involving builders, see Naming Manager Singletons on page 457 for more information about the builder provisioning mechanism.

If there is no InitialContext Factory Builder set, the class name of a class implementing the InitialContextFactory interface is specified as a property in the environment. The environment is a Hashtable object that is constructed from different sources and then merged with System properties and a resource in the calling bundle, see Environment on page 457. In a standard Java SE JNDI, the given class name is then used to construct an InitialContextFactory object and this object is then used to create the backing Context. This process is depicted in Figure 126.2 on page 456.

Figure 126.2 Backing Context

126.2.3 URL Context Factory

The InitialContext class implements the Context interface. It can therefore delegate all the Context interface methods to the backing Context object. However, it provides a special URL lookup behavior for names that are formed like URLs, that is, names that contain a colon (`:`) character. This behavior is called a URL lookup.

URL lookups are not delegated to the backing Context but are instead first tried via a URL Context based lookup on the given scheme, like:
myscheme: foo

For example a lookup using acme:foo/javax.sql.DataSource results in a URL Context being used, rather than the backing Context.

JNDI uses class loading techniques to search for an ObjectFactory class that can be used to create this URL Context. The Naming Manager provides a static method getUrlContext for this purpose. If such a URL Context is found, it is used with the requested operation and uses the full URL. If no such URL Context can be found, the backing Context is asked to perform the operation with the given name.

The URL lookup behavior is only done when the backing Context was created by the JNDI implementation in the JRE. If the backing Context had been created through the singleton provisioning mechanism, then no URL lookup is done for names that have a colon. The URL lookup responsibility is then left to the backing Context implementation.

### 126.2.4 Object and Reference Conversion

The NamingManager class provides a way to create objects from a description with the getObjectInstance method. In general, it will iterate over a number of ObjectFactory objects and ask each one of them to provide the requested object. The first non-null result indicates success. These ObjectFactory objects are created from an environment property.

A special case for the description argument in the getObjectInstance method is the Reference. A Reference is a description of an object that can be stored persistently. It can be re-created into an actual object through the static ObjectFactory method of the NamingManager class. The Reference object describes the actual ObjectFactory implementing class that must be used to create the object.

This default behavior is completely replaced with the Object Factory Builder singleton by getting the to be used ObjectFactory object directly from the set singleton Object Factory Builder.

### 126.2.5 Environment

JNDI clients need a way to set the configuration properties to select the proper JNDI Provider. For example, a JNDI Provider might require an identity and a password in order to access the service. This type of configuration is referred to as the environment of a Context. The environment is a set of properties. Common property names can be found in [JNDI Standard Property Names]. The set of properties is build from the following sources (in priority order, that is later entries are shadowed by earlier entries):

1. Properties set in the environment Hashtable object given in the constructor argument (if any) of the InitialContext class.
2. Properties from the Java System Properties
3. Properties found in $JAVA_HOME/lib/jndi.properties

There are some special rules around the handling of specific properties.

### 126.2.6 Naming Manager Singletons

The default behavior of the JRE implementation of JNDI can be extended in a standardized way. The NamingManager class has two static singletons that allow JNDI Providers outside the JRE to provide InitialContextFactory and ObjectFactory objects. These singletons are set with the following static methods on the NamingManager class:

- setObjectFactoryBuilder(ObjectFactoryBuilder) - A hook to provide ObjectFactory objects.
- setInitialContextFactoryBuilder(InitialContextFactoryBuilder) - A hook to provide InitialContextFactory objects. This hook is consulted to create a Context object that will be associated with an InitialContext object the client creates.
These JNDI Provider hooks are *singletons* and must be set before any application code creates an InitialContext object or any objects are converted. If these singletons are not set, the JNDI implementation in the JRE will provide a default behavior that is based on searching through classes defined in an environment property.

Both singletons can only be set once. A second attempt to set these singletons results in an Illegal State Exception being thrown.

### 126.2.7 Built-In JNDI Providers

The Java Runtime Environment (JRE) defines the following default providers:

- **LDAP** - Lightweight Directory Access Protocol (LDAP) service provider
- **COS** - Corba Object Service (COS) naming service provider
- **RMI** - Remote Method Invocation (RMI) Registry service provider
- **DNS** - Domain Name System (DNS) service provider

Although these are the default JNDI Service Providers, the JNDI architecture provides a number of mechanisms to plug-in new types of providers.

### 126.3 JNDI Context Manager Service

The JNDI Context Manager service allows clients to obtain a Context using the OSGi service model. By obtaining a JNDI Context Manager service, a client can get a Context object so that it can interact with the available JNDI Providers. This service replaces the approach where the creation of a new InitialContext object provided the client with access to an InitialContext object that was backed by a JNDI Provider’s Context.

The JNDIContextManager interface defines the following methods for obtaining Context objects:

- **newInitialContext()** - Obtain a Context object using the default environment properties.
- **newInitialContext(Map)** - Get a Context object using the default environment properties merged with the given properties.
- **newInitialDirContext()** - Get a DirContext object using a default environment properties.
- **newInitialDirContext(Map)** - Get a DirContext object using the default environment properties merged with the given properties.

The JNDI Context Manager service returns Context objects that implement the same behavior as the InitialContext class; the returned Context object does not actually extend the InitialContext class, its only guarantee is that it implements the Context interface.

This Context object is a facade for the context that is created by the JNDI Provider. This JNDI Provider’s Context is called the *backing Context*. This is similar to the behavior of the InitialContext class. However, in this specification, the facade can change or loose the backing Context due to the dynamics of the OSGi framework.

The returned facade must also provides URL lookups, just like an Initial Context. However, the URL Context lookup must be based on Object Factory services with a service property that defines the scheme.

The environment properties used to create the backing Context are constructed in a similar way as the environment properties of the Java SE JNDI, see *Environment and Bundles* on page 459.

The following sections define in detail how a JNDI Provider Context must be created and managed.
126.3.1 Environment and Bundles

The Java SE JNDI looks for a file in $JAVAHOME/lib/jndi.properties, see Environment on page 457. A JNDI Implementation must not use this information but it must use a resource in the bundle that uses the JNDI Context Manager service. The order is therefore:

1. Properties set in the environment Hashtable object given in the constructor argument (if any) of the InitialContext class.
2. Properties from the Java System Properties
3. A properties resource from the bundle that uses the service called /jndi.properties.

The following four properties do not overwrite other properties but are merged:

- java.naming.factory.object
- java.naming.factory.state
- java.naming.factory.control
- java.naming.factory.url.pkgs

These property values are considered lists and the ultimate value used by the JNDI Providers is taken by merging the values found in each stage into a single colon separated list. For more information see [3] JNDI Standard Property Names.

The environment consists of the merged properties. This environment is then passed to the Initial Context Factory Builder for the creation of an Initial Context Factory.

126.3.2 Context Creation

When a client calls one of the newInitialContext (or newInitialDirContext) methods, the JNDI Context Manager service must construct an object that implements the Context interface based on the environment properties. All factory methods in the InitialContextFactory and InitialContextFactoryBuilder classes take a Hashtable object with the environment as an argument, see Environment and Bundles on page 459.

The caller normally provides a specific property in the environment that specifies the class name of a provider class. This property is named:

java.naming.factory.initial

The algorithm to find the provider of the requested Context can differ depending on the presence or absence of the java.naming.factory.initial property in the environment.

In the following sections the cases for presence or absence of the java.naming.factory.initial property are described. Several steps in these algorithm iterate over a set of available services. This iteration must always take place in service ranking order. Service ranking order follows the ordering of the service.ranking service property, which is the highest service.ranking value, or when equal, the lowest service.id value.

Exception handling in the following steps is as follows:

- If an Exception is thrown by an Initial Context Factory Builder service, then this Exception must be logged but further ignored.
- Exceptions thrown by the InitialContextFactory objects when creating a Context must be thrown to the caller.

126.3.2.1 Implementation Class Present in Environment

If the implementation class is specified, a JNDI Provider is searched in the service registry with the following steps, which stop when a backing Context can be created:
1. Find a service in ranking order that has a name matching the given implementation class name as well as the InitialContextFactory class name. The searching must take place through the Bundle Context of the requesting bundle but must not require that the requesting bundle imports the package of the implementation class. If such a matching Initial Context Factory service is found, it must be used to construct the Context object that will act as the backing Context.

2. Get all the Initial Context Factory Builder services. For each such service, in ranking order:
   - Ask the Initial Context Factory Builder service to create a new InitialContextFactory object. If this is null then continue with the next service.
   - Create the Context with the found Initial Context Factory and return it.

3. If no backing Context could be found using these steps, then the JNDI Context Manager service must throw a No Initial Context Exception.

126.3.2.2 No Implementation Class Specified

If the environment does not contain a value for the java.naming.factory.initial property then the following steps must be used to find a backing Context object.

1. Get all the Initial Context Factory Builder services. For each such service, in ranking order, do:
   - Ask the Initial Context Factory Builder service to create a new InitialContextFactory object. If this is null, then continue with the next service.
   - Create the backing Context object with the found Initial Context Factory service and return it.

2. Get all the Initial Context Factory services. For each such service, in ranking order, do:
   - Ask the Initial Context Factory service to create a new Context object. If this is null then continue with the next service otherwise create a new Context with the created Context as the backing Context.

3. If no Context has been found, an initial Context is returned without any backing. This returned initial Context can then only be used to perform URL based lookups.

126.3.3 Rebinding

A JNDI Provider can be added or removed to the service registry at any time because it is an OSGi service; OSGi services are by their nature dynamic. When a JNDI Provider unregisters an Initial Context Factory that was used to create a backing service then the JNDI Context Manager service must remove the association between any returned Contexts and their now invalid backing Contexts.

The JNDI Context Manager service must try to find a replacement whenever it is accessed and no backing Context is available. However, if no such replacement can be found the called function must result in throwing a No Initial Context Exception.

126.3.4 Life Cycle and Dynamism

When a client has finished with a Context object, then the client must close this Context object by calling the close method. When a Context object is closed, the resources held by the JNDI Implementation on the client’s behalf for that Context must all be released. Releasing these resources must not affect other, independent, Context objects returned to the same client.

If a client ungets the JNDI Context Manager service, all the Context objects returned through that service instance must automatically be closed by the JNDI Context Manager. When the JNDI Context Manager service is unregistered, the JNDI Context Manager must automatically close all Contexts held.

For more information about life cycle issues, see also Life Cycle Mismatch on page 467.
126.4 JNDI Provider Admin service

JNDI provides a general object conversion service, see Object and Reference Conversion on page 457. For this specification, the responsibility of the static method on the NamingManager getInstance is replaced with the JNDI Provider Admin service. The JNDIProviderAdmin interface provides the following methods that can be used to convert a description object to an object:

- `getInstance(Object,Name,Context,Map)` - Used by Context implementations to convert a description object to another object.
- `getInstance(Object,Name,Context,Map,Attributes)` - Used by a Dir Context implementations to convert a description object to another object.

In either case, the first argument is an object, called the description. JNDI allows a number of different Java types here. When either method is called, the following algorithm is followed to find a matching Object Factory to find/create the requested object. This algorithm is identical for both methods, except that the call that takes the Attributes argument consults Dir Object Factory services first and then Object Factory services while the method without the Attributes parameter only consults Object Factory services.

1. If the description object is an instance of Referenceable, then get the corresponding Reference object and use this as the description object.
2. If the description object is not a Reference object then goto step 5.
3. If a factory class name is specified, the JNDI Provider Admin service uses its own Bundle Context to search for a service registered under the Reference’s factory class name. If a matching Object Factory is found then it is used to create the object from the Reference object and the algorithm stops here.
4. If no factory class name is specified, iterate over all the Reference object’s StringRefAddrs objects with the address type of URL. For each matching address type, use the value to find a matching URL Context, see URL Context Provider on page 463, and use it to recreate the object. See the Naming Manager for details. If an object is created then it is returned and the algorithm stops here.
5. Iterate over the Object Factory Builder services in ranking order. Attempt to use each such service to create an ObjectFactory or DirObjectFactory instance. If this succeeds (non null) then use this ObjectFactory or DirObjectFactory instance to recreate the object. If successful, the algorithm stops here.
6. If the description was a Reference and without a factory class name specified, or if the description was not of type Reference, then attempt to convert the object with each Object Factory service (or Dir Object Factory service for directories) service in ranking order until a non-null value is returned.
7. If no ObjectFactory implementations can be located to resolve the given description object, the description object is returned.

If an Exception occurs during the use of an Object Factory Builder service then this exception should be logged but must be ignored. If, however, an Exception occurs during the calling of a found ObjectFactory or DirObjectFactory object then this Exception must be re-thrown to the caller of the JNDI Provider Admin service.

126.5 JNDI Providers

JNDI Providers can be registered by registering an appropriate service. These services are consulted by the JNDI Implementation for creating a Context as well as creating/finding/convert general objects.
126.5.1 Initial Context Factory Builder Provider

An Initial Context Factory Builder provider is asked to provide an Initial Context Factory when no implementation class is specified or no such implementation can be found. An Initial Context Factory Builder service can be used by containers for other bundles to control the initial Context their applications receive.

An Initial Context Factory Builder provider must register an Initial Context Factory Builder service. The service.ranking property defines the iteration ordering of multiple Initial Context Factory Builder services. Implementations must be careful to correctly provide defaults.

For example, a container could use a thread local variable to mark the stack for a specific application. The implementation of the Initial Context Factory Builder can then detect specific calls from this application. To make the next code example work, an instance must be registered as an Initial Context Factory Builder service.

```java
public class Container implements InitialContextFactoryBuilder {
    ThreadLocal<Application> apps;
    void startApp(final Application app) {
        Thread appThread = new Thread(app.getName()) {
            public void run() {
                apps.set(app);
                app.run();
            }
        }
    }
    public InitialContextFactory createInitialContextFactory( Hashtable<?,?> ht ) {
        final Application app = apps.get();
        if ( app == null )
            return null;
        return new InitialContextFactory() {
            public Context getInitialContext( Hashtable<?,?>env) {
                return app.getContext(env);
            }
        };
    }
}
```

126.5.2 Initial Context Factory Provider

An Initial Context Factory provides Contexts of a specific type. For example, those contexts allow communications with an LDAP server. An Initial Context Factory Provider must register the its Initial Context Factory service under the following names:

- Implementation Class - An Initial Context Factory provider must register a service under the name of the implementation class. This allows the JNDI Context Manager to find implementations specified in the environment properties.
- Initial Context Factory - As a general Initial Context Factory. If registered as such, it can be consulted for a default Initial Context. Implementations must be careful to only return a Context when the environment properties are appropriate. See No Implementation Class Specified on page 460

An Initial Context Factory service can create both DirContext as well as Context objects.

For example, SUN JREs for Java SE provide an implementation of a Context that can answer DNS questions. The name of the implementation class is a well known constant. The following class can be used with Declarative Services to provide a lazy implementation of a DNS Context:
public class DNSProvider implements InitialContextFactory{
    public Context createInitialContextFactory( Hashtable<?,?>env ) throws
        NamingException {
        try {
            Class<InitialContextFactory> cf = (Class<InitialContextFactory>)
            l.loadClass("com.sun.jndi.dns.DnsContextFactory" );
            InitialContextFactory icf = cf.newInstance();
            return icf.createInitialContextFactory(env);
        }
        catch( Throwable t ) { return null; }
    }
}

126.5.3 Object Factory Builder Provider

An Object Factory Builder provider must register an Object Factory Builder service. Such a service can be used to provide ObjectFactory and/or DirObjectFactory objects. An Object Factory Builder service is requested for such an object when no specific converter can be found. This service can be leveraged by bundles that act as a container for other bundles to control the object conversion for their subjects.

126.5.4 Object Factory Provider

An Object Factory provider can participate in the conversion of objects. It must register a service under the following names:

- **Implementation Class** - A service registered under its implementation class can be leveraged by a description that is a Reference object. Such an object can contain the name of the factory class. The implementation class can implement the DirObjectFactory interface or the ObjectFactory interface.
- **Object Factory** - The ObjectFactory interface is necessary to ensure class space consistency.
- **Dir Object Factory** - If the Object Factory provider can accept the additional Attributes argument in the getObjectInstance method of the JNDI Provider Admin service than it must also register as a Dir Object Factory service.

126.5.5 URL Context Provider

A **URL Context Factory** is a special type of an Object Factory service. A URL Context Factory must be registered as an Object Factory service with the following service property:

**osgi.jndi.url.scheme** - The URL scheme associated with this URL Context, for example acme. The scheme must not contain the colon (": :").

A URL Context is used for URL based operations on an initial Context. For example, a lookup to acme:foo/javax.sql.DataSource must not use the provider based lookup mechanism of the backing Context but instead causes a lookup for the requested URL Context. A URL Context also provides a secondary mechanism for restoring Reference objects.

When an initial Context returned by the JNDI Context Manager service is given a URL based operation, it searches in the service registry for an Object Factory service that is published with the URL scheme property that matches the scheme used from the lookup request.

It then calls the getInstance method on the Object Factory service with the following parameters:

- **Object** - Should be either a String, String[], or null.
- **Name** - must be null
- **Context** - must be null
Hashtable - The environment properties.

Calling the getInstance method must return a Context object. This context is then used to perform the lookup.

The life cycle of the Object Factory used to create the URL Context is tied to the JNDI context that was used to perform the URL based JNDI operation. By the time JNDI context is closed any ObjectFactory objects held to process the URL lookups must be released (unget).

### 126.5.6 JRE Context Providers

The Java Runtime Environment (JRE) defines a number of default naming providers, see Built-In JNDI Providers on page 458. These naming providers are not OSGi aware, but are commonly used and are provided by the JRE. These naming providers rely on the NamingManager class for object conversion and finding URL Contexts.

The JRE default providers are made available by the JNDI Implementation. This JNDI Implementation must register a built-in Initial Context Factory Builder service that is capable of loading any InitialContextFactory classes of the JRE providers.

When this built-in Initial Context Factory Builder is called to create an InitialContextFactory object it must look in the environment properties that were given as an argument and extract the java.naming.factory.initial property; this property contains the name of the class of a provider. The built-in Initial Context Factory Builder then must use the bootstrap class loader to load the given InitialContextFactory class and creates a new instance with the no arguments constructor and return it. If this fails, it must return null. This mechanism will allow loading of any built-in providers.

This built-in Initial Context Factory Builder service must be registered with no service.ranking property. This will give it the default ranking and allows other providers to override the default.

### 126.6 OSGi URL Scheme

A URL scheme is available that allows JNDI based applications to access services in the service registry, see Services and State on page 466 about restrictions on these services. The URL scheme is specified as follows:

```
service ::= 'osgi:service/' query
query ::= jndi-name | qname ('/' filter )?
jndi-name ::= <any string>
```

No spaces are allowed between the terms.

This OSGi URL scheme can be used to perform a lookup of a single matching service using the interface name and filter. The URL Context must use the owning bundle to perform the service queries. The owning bundle is the bundle that requested the initial Context from the JNDI Context Manager service or received its Context through the InitialContext class. The returned objects must not be incompatible with the class space of the owning bundle.

The lookup for a URL with the osgi: scheme and service path returns the service with highest service.ranking and the lowest service.id. This scheme only allows a single service to be found. Multiple services can be obtained with the osgi: scheme and servicelist path:

```
servicelist ::= 'osgi:servicelist/' query?
```

If this osgi:servicelist scheme is used from a lookup method then a Context object is returned instead of a service object. Calling the listBindings method will produce a NamingEnumeration object that provides Binding objects. A Binding object contains the name, class of the service, and the service object. The bound object is the service object contained in the given Context.
When the Context class list method is called, the Naming Enumeration object provides a NameClassPair object. This NameClassPair object will include the name and class of each service in the Context. The list method can be useful in cases where a client wishes to iterate over the available services without actually getting them. If the service itself is required, then listBindings method should be used.

If multiple services matched the criteria listed in the URL, there would be more than one service available in the Context, and the corresponding Naming Enumeration would contain the same number of services.

If multiple services match, a call to listBindings on this Context would return a list of bindings whose name are a string with the service.id number, for example:

1283

Thus the following lookup is valid:

osgi:servicelist/javax.sql.DataSource/(&{db=mydb}{version=3.1})

A service can provide a JNDI service name if it provides the following service property:

- osgi.jndi.service.name - An alternative name that the service can be looked up by when the osgi: URL scheme is used.

If a service is published with a JNDI service name then the service matches any URL that has this service name in the place of interface. For example, if the JNDI service name is foo, then the following URL selects this service:

osgi:service/foo

Using a JNDI service name that can be interpreted as an interface name must be avoided, if this happens the result is undefined.

A JNDI client can also obtain the Bundle Context of the owning bundle by using the osgi: namespace with the framework/bundleContext name. The following URL must return the Bundle Context of the owning bundle:

osgi:framework/bundleContext

After the NamingEnumeration object has been used it must be closed by the client. Implementations must then unget any gotten services or perform other cleanup.

126.6.1 Service Proxies

The OSGi URL Context handles the complexities by hiding the dynamic nature of OSGi. The OSGi URL Context must handle the dynamics by proxying the service objects. This proxy must implement the interface given in the URL. If the JNDI service name instead of a class name is used, then all interfaces under which the service is registered must be implemented. If an interface is not compatible with the owning bundle's class space then it must not be implemented on the proxy, it must then be ignored. If this results in no implemented interfaces then an Illegal Argument Exception must be thrown.

Interfaces can always be proxied but classes are much harder. For this reason, an implementation is free to throw an Illegal Argument Exception when a class is used in the URL or in one of the registration names.

Getting the actual service object can be delayed until the proxy is actually used to call a method. If a method is called and the actual service has been unregistered, then the OSGi URL Context must attempt to rebind it to another service that matches the criteria given in the URL the next time it is
called. When no alternative service is available, a Service Exception with the UNREGISTERED type code must be thrown. Services obtained with the osgi: URL scheme must therefore be stateless because the re-binding to alternative services is not visible to the caller; there are no listeners defined for this re-binding, see Services and State on page 466.

If the reference was looked up using osgi:servicelist then proxies must still be used, however, these proxies must not rebind when their underlying service is unregistered. Instead, they must throw a Service Exception with the UNREGISTERED type whenever the proxy is used and the proxied service is no longer available.

### 126.6.2 Services and State

A service obtained through a URL Context lookup is proxied. During the usage of this service, the JNDI Implementation can be forced to transparently rebind this service to another instance. The JNDI specification is largely intended for portability. For this reason, it has no mechanism architect-ed to receive notifications about this rebinding. The client code is therefore unable to handle the dynamics.

The consequence of this model is that stateful services require extra care because applications cannot rely on the fact that they always communicate with the same service. Virtually all OSGi specified services have state.

### 126.7 Traditional Client Model

A JNDI Implementation must at startup register the InitialContextFactoryBuilder object and the ObjectFactoryBuilder object with the NamingManager class. As described in JNDI Overview on page 456, the JNDI code in the JRE will then delegate all Context related requests to the JNDI Implementation. Setting these singletons allows code that is not aware of the OSGi framework to use Context implementations from JNDI Providers registered with the OSGi service registry and that are managed as bundles. The JNDI Implementation therefore acts as a broker to the service registry for OSGi unaware code.

This brokering role can only be played when the JNDI Implementation can set the singletons as specified in Naming Manager Singletons on page 457. If the JNDI Implementation cannot set these singletons then it should log an error with the Log Service, if available. It can then not perform the following sections.

### 126.7.1 New Initial Context

The client typically requests a Context using the following code:

```java
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
InitialContext ctx = new InitialContext(env);
```

The created InitialContext object is a facade for the real Context that is requested by the caller. It provides the bootstrapping mechanism for JNDI Provider plugability. In order to obtain the provider's Context, the InitialContext class makes a call to the static getContext method on the NamingManager class. The JNDI code in the JRE then delegates any request for an initial Context object to the JNDI Implementation through the registered InitialContextFactoryBuilder singleton. The JNDI Implementation then determines the Bundle Context of the caller as described in Caller's Bundle Context on page 467. If no such Bundle Context can be found, a No Initial Context Exception is thrown to the caller. This Bundle Context must be from an ACTIVE bundle.

This Bundle Context is then used to get the JNDI Context Manager service. This service is then used as described in Context Creation on page 459 to get an initial Context. This initial Context is then used in the InitialContext object as the default initial context. In this specification this is normally
called the backing context. An InitialContext object constructed through an Initial Context Factory Builder will not use the URL lookup mechanism, it must delegate all operations to the its backing context. A Context obtained through the JNDI Context Manager provides the URL lookup behavior instead.

126.7.2 Static Conversion

JNDI provides a general object conversion facility that is used by the URL Context and the process of restoring an object from a Reference object, see Object and Reference Conversion on page 457. A JNDI Implementation must take over this conversion by setting the static Object Factory Builder singleton, see Naming Manager Singletons on page 457. Non-OSGi aware Context implementations will use the NamingManager static getObjectName method for object conversion. This method then delegates to the set singleton Object Factory Builder to obtain an ObjectFactory object that understands how to convert the given description to an object. The JNDI Implementation must return an Object Factory that understands the OSGi service registry. If the getObjectName method is called on this object it must use the same rules as defined for the JNDI Provider Admin service

126.7.3 Caller’s Bundle Context

The following mechanisms are used to determine the callers Bundle Context:

1. Look in the JNDI environment properties for a property called

   osgi.service.jndi.bundleContext

   If a value for this property exists then use it as the Bundle Context. If the Bundle Context has been found stop.

2. Obtain the Thread Context Class Loader; if it, or an ancestor class loader, implements the BundleReference interface, call its getBundle method to get the clients Bundle; then call getBundleContext on the Bundle object to get the clients Bundle Context. If the Bundle Context has been found stop.

3. Walk the call stack until the invoker is found. The invoker can be the caller of the InitialContext class constructor or the NamingManager or DirectoryManager getObjectName methods.
   - Get the class loader of the caller and see if it, or an ancestor, implements the BundleReference interface.
   - If a Class Loader implementing the BundleReference interface is found call the getBundle method to get the clients Bundle; then call the getBundleContext method on the Bundle to get the clients Bundle Context.
   - If the Bundle Context has been found stop, else continue with the next stack frame.

126.7.4 Life Cycle Mismatch

The use of static access to the JNDI mechanisms, NamingManager and InitialContext class methods, in the traditional client programming model produces several problems with regard to the OSGi life cycle. The primary problem being that there is no dependency management in place when static methods are used. These problems do not exist for the JNDI Context Manager service. Therefore, OSGi applications are strongly encouraged to use the JNDI Context Manager service.

The traditional programming model approach relies on two JVM singletons in the Naming Manager, see Naming Manager Singletons on page 457. The JNDI Implementation bundle must set both singletons before it registers its JNDI Context Manager service and JNDI Provider Admin service. However, in OSGi there is no defined start ordering, primarily because bundles can be updated at
Security

126.8 Security

126.8.1 JNDI Implementation

A JNDI Implementation may wish to assert that the user of the provider has some relevant Java 2 security permission. Since the JNDI implementation is an intermediary between the JNDI client and provider this means that the JNDI implementation needs to have any permissions required to access any JNDI Provider. As a result the JNDI implementation needs All Permission. This will result in the JNDI clients permissions being checked to see if it has the relevant permission to access the JNDI Provider.

The JNDI Implementation must make any invocation to access these services in a doPrivileged check. A JNDI client must therefore not be required to have the following permissions, which are needed by a JNDI Implementation:

- ServicePermission ..ObjectFactory: REGISTER, GET
- ServicePermission ..DirObjectFactory: REGISTER, GET
- ServicePermission ..ObjectFactoryBuilder: REGISTER, GET
- ServicePermission ..InitialContextFactory: REGISTER, GET
- ServicePermission ..InitialContextFactoryBuilder: REGISTER, GET
- ServicePermission ..JNDIProviderAdmin: REGISTER, GET

The JNDI Implementation bundle must have the appropriate permissions to install the InitialContextFactoryBuilder and ObjectFactoryBuilder instances using the appropriate methods on the NamingManager class. This requires the following permission:

- RuntimePermission "setFactory"

126.8.2 JNDI Clients

A JNDI client using the JNDI Context Manager service must have the following permissions:

- ServicePermission ..JNDIContextManager: GET

Obtaining a reference to a JNDI Context Manager service should be considered a privileged operation and should be guarded by permissions.

126.8.3 OSGi URL namespace

A JNDI client must not be able to obtain services or a Bundle Context that the client bundle would not be able to get via the core OSGI API. To allow a client to use the osgi namespace to get a service the bundle must have the corresponding Service Permission. When using the osgi namespace to obtain the Bundle Context the client bundle must have Admin Permission for the Bundle Context.
These permissions must be enforced by the osgi URL namespace handler. If there is no proper permission, the implementation must throw a Name Not Found Exception to prevent exposing the existence of such services.

126.9  org.osgi.service.jndi

JNDI 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.jndi; version="[1.0,2.0)"

Example import for providers implementing the API in this package:
Import-Package: org.osgi.service.jndi; version="[1.0,1.1)"

126.9.1 Summary

- JNDIConstants - Constants for the JNDI implementation.
- JNDIContextManager - This interface defines the OSGi service interface for the JNDIContextManager.
- JNDIProviderAdmin - This interface defines the OSGi service interface for the JNDIProviderAdmin service.

126.9.2 public class JNDIConstants

Constants for the JNDI implementation.

Concurrenty  Immutable

126.9.2.1 public static final String BUNDLE_CONTEXT = "osgi.service.jndi.bundleContext"

This JNDI environment property can be used by a JNDI client to indicate the caller’s BundleContext. This property can be set and passed to an InitialContext constructor. This property is only useful in the "traditional" mode of JNDI.

126.9.2.2 public static final String JNDI_SERVICENAME = "osgi.jndi.service.name"

This service property is set on an OSGi service to provide a name that can be used to locate the service other than the service interface name.

126.9.2.3 public static final String JNDI_URLSCHEME = "osgi.jndi.url.scheme"

This service property is set by JNDI Providers that publish URL Context Factories as OSGi Services. The value of this property should be the URL scheme that is supported by the published service.

126.9.3 public interface JNDIContextManager

This interface defines the OSGi service interface for the JNDIContextManager. This service provides the ability to create new JNDI Context instances without relying on the InitialContext constructor.

Concurrenty  Thread-safe

126.9.3.1 public Context newInitialContext() throws NamingException

- Creates a new JNDI initial context with the default JNDI environment properties.
126.9.3.2 public Context newInitialContext(Map environment) throws NamingException

- **environment** JNDI environment properties specified by caller
  - Creates a new JNDI initial context with the specified JNDI environment properties.

- **Returns** an instance of javax.naming.Context
- **Throws** NamingException – upon any error that occurs during context creation

126.9.3.3 public DirContext newInitialDirContext() throws NamingException

- Creates a new initial DirContext with the default JNDI environment properties.

- **Returns** an instance of javax.naming.directory.DirContext
- **Throws** NamingException – upon any error that occurs during context creation

126.9.3.4 public DirContext newInitialDirContext(Map environment) throws NamingException

- **environment** JNDI environment properties specified by the caller
  - Creates a new initial DirContext with the specified JNDI environment properties.

- **Returns** an instance of javax.naming.directory.DirContext
- **Throws** NamingException – upon any error that occurs during context creation

### 126.9.4 JNDIProviderAdmin Interface

This interface defines the OSGi service interface for the JNDIProviderAdmin service. This service provides the ability to resolve JNDI References in a dynamic fashion that does not require calls to `NamingManager.getObjectInstance()`. The methods of this service provide similar reference resolution, but rely on the OSGi Service Registry in order to find `ObjectFactory` instances that can convert a Reference to an Object. This service will typically be used by OSGi-aware JNDI Service Providers.

- **Concurrency** Thread-safe

#### 126.9.4.1 public Object getObjectInstance(Object refInfo, Name name, Context context, Map environment) throws Exception

- **refInfo** Reference info
- **name** the JNDI name associated with this reference
- **context** the JNDI context associated with this reference
- **environment** the JNDI environment associated with this JNDI context
  - Resolve the object from the given reference.

- **Returns** an Object based on the reference passed in, or the original reference object if the reference could not be resolved.
- **Throws** Exception – in the event that an error occurs while attempting to resolve the JNDI reference.

#### 126.9.4.2 public Object getObjectInstance(Object refInfo, Name name, Context context, Map environment, Attributes attributes) throws Exception

- **refInfo** Reference info
- **name** the JNDI name associated with this reference
- **context** the JNDI context associated with this reference
- **environment** the JNDI environment associated with this JNDI context
attributes  the naming attributes to use when resolving this object

- Resolve the object from the given reference.

Returns  an Object based on the reference passed in, or the original reference object if the reference could not be resolved.

Throws  Exception – in the event that an error occurs while attempting to resolve the JNDI reference.

126.10  References

[1]  Java Naming and Directory Interface
http://docs.oracle.com/javase/6/docs/technotes/guides/jndi/index.html

http://download.oracle.com/javase/6/docs/technotes/guides/jndi/index.html

http://download.oracle.com/javase/1.5.0/docs/api/javax/naming/Context.html
127 JPA Service Specification

Version 1.0

127.1 Introduction

The Java Persistence API (JPA) is a specification that sets a standard for persistently storing objects in enterprise and non-enterprise Java based environments. JPA provides an Object Relational Mapping (ORM) model that is configured through persistence descriptors. This Java Persistence Service specification defines how persistence units can be published in an OSGi framework, how client bundles can find these persistence units, how database drivers are found with the OSGi JDBC Specification, as well as how JPA providers can be made available within an OSGi framework.

Applications can be managed or they can be unmanaged. Managed applications run inside a Java EE Container and unmanaged applications run in a Java SE environment. The managed case requires a provider interface that can be used by the container, while in the unmanaged case the JPA provider is responsible for supporting the client directly. This specification is about the unmanaged model of JPA except in the areas where the managed model is explicitly mentioned. Additionally, multiple concurrent providers for the unmanaged case are not supported.

This JPA Specification supports both [1] JPA 1.0 and [2] JPA 2.0.

127.1.1 Essentials

- **Dependencies** - There must be a way for persistence clients, if they so require, to manage their dependencies on a compatible persistence unit.
- **Compatibility** - The Persistence Unit service must be able to function in non-managed mode according to existing standards and interfaces outlined in the JPA specification.
- **Modularity** - Persistent classes and their accompanying configuration can exist in a separate bundle from the client that is operating on them using the Persistence Unit service.
- **JDBC** - Leverage the OSGi JDBC Specification for access to the database.

127.1.2 Entities

- **JPA Provider** - An implementation of JPA, providing the Persistence Provider and JPA Services to Java EE Containers and Client Bundles.
- **Interface Bundle** - A bundle containing the interfaces and classes in the javax.persistence namespace (and its sub-namespaces) that are defined by the JPA specification.
- **Persistence Bundle** - A bundle that includes, a Meta-Persistence header, one or more Persistence Descriptor resources, and the entity classes specified by the Persistence Units in those resources.
- **Client Bundle** - The bundle that uses the Persistence Bundle to retrieve and store objects.
- **Persistence Descriptor** - A resource describing one or more Persistence Units.
- **Persistence Unit** - A named configuration for the object-relational mappings and database access as defined in a Persistence Descriptor.
- **EntityManager** - The interface that provides the control point of retrieving and persisting objects in a relational database based on a single Persistence Unit for a single session.
127.1.3 Dependencies

This specification is based on JPA 1.0 and JPA 2.0. JPA 2.0 is backward compatible with JPA 1.0. For this reason, the versions of the packages follow the OSGi recommended version policy with the addition of a special JPA marker that annotates the specification version for JPA. All JPA Packages must also have an attribute called \textit{jpa} that specifies the JPA version. The purpose of this attribute is to make it clear what JPA version belongs to this package.

<table>
<thead>
<tr>
<th>JPA Packages</th>
<th>Export Version</th>
<th>Client Import Range</th>
<th>Provider Imp. Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.persistence</td>
<td>1.0</td>
<td>[1.0,2.0)</td>
<td>[1.0,1.1)</td>
</tr>
<tr>
<td>javax.persistence.spi</td>
<td>1.0</td>
<td>[1.0,2.0)</td>
<td>[1.0,1.1)</td>
</tr>
</tbody>
</table>

Figure 127.1 JPA Service overview
### JPA Packages

<table>
<thead>
<tr>
<th>JPA</th>
<th>Packages</th>
<th>Export Version</th>
<th>Client Import Range</th>
<th>Provider Imp. Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPA 2.0</td>
<td>javax.persistence</td>
<td>1.1</td>
<td>[1.1,2.0)</td>
<td>[1.1,1.2)</td>
</tr>
<tr>
<td></td>
<td>javax.persistence.spi</td>
<td>1.1</td>
<td>[1.1,2.0)</td>
<td>[1.1,1.2)</td>
</tr>
</tbody>
</table>

For example, JPA should have an export declaration like:

```
Export-Package: javax.persistence; version=1.1; jpa=2.0, ...
```

### Synopsis

A JPA Provider tracks Persistence Bundles; a Persistence Bundle contains a Meta-Persistence manifest header. This manifest header enumerates the Persistence Descriptor resources in the Persistence Bundle. Each resource's XML schema is defined by the JPA 1.0 or JPA 2.0 specification. The JPA Provider reads the resource accordingly and extracts the information for one or more Persistence Units. For each found Persistence Unit, the JPA Provider registers an Entity Manager Factory Builder service. If the database is defined in the Persistence Unit, then the JPA Provider registers an Entity Manager Factory service during the availability of the corresponding Data Source Factory.

The identification of these services is handled through a number of service properties. The Entity Manager Factory service is named by the standard JPA interface, the Builder version is OSGi specific; it is used when the Client Bundle needs to create an Entity Manager Factory based on configuration properties.

A Client Bundle that wants to persist or retrieve its entity classes depends on an Entity Manager Factory (Builder) service that corresponds to a Persistence Unit that lists the entity classes. If such a service is available, the client can use this service to get an Entity Manager, allowing the client to retrieve and persist objects as long as the originating Entity Manager Factory (Builder) service is registered.

In a non-OSGi environment, it is customary to get an Entity Manager Factory through the Persistence class. This Persistence class provides a number of static methods that give access to any locally available JPA providers. This approach is not recommended in an OSGi environment due to class loading and start ordering issues. However, OSGi environments can support access through this static factory with a Static Persistence bundle.

### JPA Overview

Java Persistence API (JPA) is a specification that is part of [3] Java EE 5. This OSGi Specification is based on [1] JPA 1.0 and [2] JPA 2.0. This section provides an overview of JPA as specified in the JCP.

The purpose of this section is to introduce the concepts behind JPA and define the terminology that will be used in the remainder of the chapter.

The purpose of JPA is to simplify access to relational databases for applications on the object-oriented Java platform. JPA provides support for storing and retrieving objects in a relational database. The JPA specification defines in detail how objects are mapped to tables and columns under the full control of the application. The core classes involved are depicted in Figure 127.2.
The JPA specifications define a number of concepts that are defined in this section for the purpose of this OSGi specification. However, the full syntax and semantics are defined in the JPA specifications.

### 127.2.1 Persistence

Classes that are stored and retrieved through JPA are called the *entity classes*. In this specification, the concept of entity classes includes the *embeddable* classes, which are classes that do not have any persistent identity, and mapped super classes that allow mappings, but are not themselves persistent. Entity classes are not required to implement any interface or extend a specific superclass, they are Plain Old Java Objects (POJOs). It is the responsibility of the JPA Provider to connect to a database and map the store and retrieve operations of the entity classes to their tables and columns. For performance reasons, the entity classes are sometimes **enhanced**. This enhancement can take place during build time, deploy time, or during class loading time. Some enhancements use byte code weaving, some enhancements are based on subclassing.

The JPA Provider cannot automatically perform its persistence tasks; it requires configuration information. This configuration information is stored in the Persistence Descriptor. A Persistence Descriptor is an XML file according of one of the two following namespaces:

http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd

The JPA standard Persistence Descriptor must be stored in META-INF/persistence.xml. It is usually in the same class path entry (like a JAR or directory) as the entity classes.

The JPA Provider parses the Persistence Descriptor and extracts one or more Persistence Units. A Persistence Unit includes the following aspects:

- **Name** - Every Persistence Unit must have a name to identify it to clients. For example: **Accounting**.
- **Provider Selection** - Restriction to a specific JPA Provider, usually because there are dependencies in the application code on provider specific functionality.
- **JDBC Driver Selection** - Selects the JDBC driver, the principal and the credentials for selecting and accessing a relational database. See **JDBC Access in JPA** on page 478.
- **Properties** - Standard and JPA Provider specific properties.

The object-relational mappings are stored in special mapping resources or are specified in annotations.

A Persistence Unit can be *complete* or *incomplete*. A complete Persistence Unit identifies the database driver that is needed for the Persistence Unit, though it does not have to contain the credentials. An incomplete Persistence Unit lacks this information.
The relations between the class path, its entries, the entity classes, the Persistence Descriptor and the Persistence Unit is depicted in Figure 127.3 on page 477.

**Figure 127.3 JPA Configuration**

JPA recognizes the concept of a *persistence root*. The persistence root is the root of the JAR (or directory) on the class path that contains the META-INF/persistence.xml resource.

### 127.2.2 JPA Provider

The JPA specifications provide support for multiple JPA Providers in the same application. An Application selects a JPA Provider through the Persistence class, using static factory methods. One of these methods accepts a map with *configuration properties*. Configuration properties can override information specified in a Persistence Unit or these properties add new information to the Persistence Unit.

The default implementation of the Persistence class discovers providers through the Java EE services model, this model requires a text resource in the class path entry called:

META-INF/servicesjavax.persistence.PersistenceProvider

This text resource contains the name of the JPA Provider implementation class.

The Persistence class `createEntityManagerFactory` method provides the JPA Provider with the name of a Persistence Unit. The JPA Provider must then scan the class path for any META-INF/persistence.xml entries, these are the available Persistence Descriptors. It then extracts the Persistence Units to find the requested Persistence Unit. If no such Persistence Unit can be found, or the JPA Provider is restricted from servicing this Persistence Unit, then null is returned. The Persistence class will then continue to try the next found or registered JPA Provider.

A Persistence Unit can restrict JPA Providers by specifying a *JPA Provider class*, this introduces a *provider dependency*. The specified JPA Provider class must implement the PersistenceProvider interface. This *implementation class name* must be available from the JPA Provider’s documentation. JPA Providers that do not own the specified JPA Provider class must ignore such a Persistence Unit.

Otherwise, if the Persistence Unit is not restricted, the JPA Provider is *assigned* to this Persistence Unit; it must be ready to provide an EntityManagerFactory object when the application requests one.

The JPA Provider uses the Persistence Unit, together with any additional configuration properties, to construct an *EntityManager Factory*. The application then uses this Entity Manager Factory to construct an *Entity Manager*, optionally providing additional configuration properties. The Entity Manager then provides the operations for the application to store and retrieve entity classes from the database.

The additional configuration properties provided with the creation of the Entity Manager Factory or the Entity Manager are often used to specify the database driver and the credentials. This allows the
Persistence Unit to be specified without committing to a specific database, leaving the choice to the application at runtime.

The relations between the application, Entity Manager, Entity Manager Factory and the JPA Provider are depicted in Figure 127.4 on page 478.

**Figure 127.4  JPA Dynamic Model**

![JPA Dynamic Model Diagram]

### 127.2.3 Managed and Unmanaged

The JPA specifications make a distinction between a *managed* and an *unmanaged* mode. In the managed mode the presence of a Java EE Container is assumed. Such a container provides many services for its contained applications like transaction handling, dependency injection, etc. One of these aspects can be the interface to the relational database. The JPA specifications therefore have defined a special method for Java EE Containers to manage the persistence aspects of their Managed Clients. This method is the `createContainerEntityManagerFactory` method on the `PersistenceProvider` interface. This method is purely intended for Java EE Containers and should not be used in other environments.

The other method on the `PersistenceProvider` interface is intended to be used by the Persistence class static factory methods. The Persistence class searches for an appropriate JPA Provider by asking all available JPA Providers to create an Entity Manager Factory based on configuration properties. The first JPA Provider that is capable of providing an Entity Manager Factory wins. The use of these static factory methods is called the *unmanaged mode*. It requires a JPA Provider to scan the class path to find the assigned Persistence Units.

### 127.2.4 JDBC Access in JPA

A Persistence Unit is configured to work with a relational database. JPA Providers communicate with a relational database through compliant JDBC database drivers. The database and driver parameters are specified in the Persistence Unit or configured during Entity Manager Factory or Entity Manager creation with the configuration properties. The configuration properties for selecting a database in non-managed mode were proprietary in JPA 1.0 but have been standardized in version 2.0 of JPA:

- `javax.persistence.jdbc.driver` - Fully-qualified name of the driver class
- `javax.persistence.jdbc.url` - Driver-specific URL to indicate database information
- `javax.persistence.jdbc.user` - User name to use when obtaining connections
- `javax.persistence.jdbc.password` - Password to use when obtaining connections
127.3 Bundles with Persistence

The primary goal of this specification is to simplify the programming model for bundles that need persistence. In this specification there are two application roles:

- **Persistence Bundle** - A Persistence Bundle contains the entity classes and one or more Persistence Descriptors, each providing one or more Persistence Units.
- **Client Bundle** - A Client Bundle contains the code that manipulates the entity classes and uses an Entity Manager to store and retrieve these entity classes with a relational database. The Client Bundle obtains the required Entity Manager(s) via a service based model.

These roles can be combined in a single bundle.

127.3.1 Services

A JPA Provider uses Persistence Units to provide Client Bundles with a configured *EntityManager Factory* service and/or an *EntityManager Factory Builder* service for each assigned Persistence Unit:

- **EntityManager Factory service** - Provides an EntityManagerFactory object that depends on a complete Persistence Unit. That is, it is associated with a registered Data Source Factory service.
- **EntityManager Factory Builder service** - The Entity Manager Factory Builder service provides the capability of creating an EntityManagerFactory object with additional configuration properties.

These services are collectively called the *JPA Services*. Entity Managers obtained from such JPA Services can only be used to operate on entity classes associated with their corresponding Persistence Unit.

127.3.2 Persistence Bundle

A *Persistence Bundle* is a bundle that specifies the Meta-Persistence header, see *Meta Persistence Header* on page 482. This header refers to one or more Persistence Descriptors in the Persistence Bundle. Commonly, this is the META-INF/persistence.xml resource. This location is the standard for non-OSGi environments, however an OSGi bundle can also use other locations as well as multiple resources.

For example, the contents of a simple Persistence Bundle with a single Person entity class could look like:

META-INF/
META-INF/MANIFEST.MF
OSGI-INF/address.xml
com/acme/Person.class

The corresponding manifest would then look like:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Meta-Persistence: OSGI-INF/address.xml
Bundle-SymbolicName: com.acme.simple.persistence
Bundle-Version: 3.2.4.200912231004
```

A Persistence Bundle is a normal bundle; it must follow all the rules of OSGi and can use all OSGi constructs like Bundle-Classpath, fragment bundles, import packages, export packages, etc. However, there is one limitation: any entity classes must originate in the bundle’s JAR, it cannot come from a fragment. This requirement is necessary to simplify enhancing entity classes.
127.3.3 Client Bundles

A Client Bundle uses the entity classes from a Persistence Bundle to provide its required functionality. To store and retrieve these entity classes a Client Bundle requires an Entity Manager that is configured for the corresponding Persistence Unit.

An Entity Manager is intended to be used by a single session, it is not thread safe. Therefore, a client needs an Entity Manager Factory to create an Entity Manager. In an OSGi environment, there are multiple routes to obtain an Entity Manager Factory.

A JPA Provider must register an Entity Manager Factory service for each assigned Persistence Unit that is complete. Complete means that it is a configured Persistence Unit, including the reference to the relational database. The Entity Manager Factory service is therefore bound to a Data Source Factory service and Client Bundles should not attempt to rebind the Data Source Factory with the configuration properties of the `createEntityManager(Map)` method. See Rebinding on page 486 for the consequences. If the Data Source Factory must be bound by the Client Bundle then the Client Bundle should use the Custom Configured Entity Manager on page 480.

The Entity Manager Factory service must be registered with the service properties as defined in Service Registrations on page 483. These are:

- `osgi.unit.name` (String) The name of the Persistence Unit
- `osgi.unit.version` (String) The version of the associated Persistence Bundle
- `osgi.unit.provider` (String) The implementation class name of the JPA Provider

The life cycle of the Entity Manager Factory service is bound to the Persistence Bundle, the JPA Provider, and the selected Data Source Factory service.

A Client Bundle that wants to use an Entity Manager Factory service should therefore use an appropriate filter to select the Entity Manager Factory service that corresponds to its required Persistence Unit. For example, the following snippet uses Declarative Services, see Declarative Services Specification on page 167, to statically depend on such a service:

```xml
<reference name="accounting" target="(&(osgi.unit.name=Accounting)(osgi.unit.version=3.2.*))" interface="javax.persistence.EntityManagerFactory"/>
```

127.3.4 Custom Configured Entity Manager

If a Client Bundle needs to provide configuration properties for the creation of an Entity Manager Factory it should use the `EntityManagerFactoryBuilder` service. This can for example be used to provide the database selection properties when the Persistence Unit is incomplete or if the database selection needs to be overridden.

The Entity Manager Factory Builder service’s life cycle must not depend on the availability of any Data Source Factory, even if a JDBC driver class name is specified in the Persistence Descriptor. The Entity Manager Factory Builder service is registered with the same service properties as the corresponding Entity Factory service, see Service Registrations on page 483.

The following method is defined on the `EntityManagerFactoryBuilder` interface:

- `createEntityManagerFactory(Map)` - Returns a custom configured `EntityManagerFactory` instance for the Persistence Unit associated with the service. Accepts a map with the configuration properties to be applied during Entity Manager Factory creation. The method must return a proper Entity Manager Factory or throw an Exception.

The `createEntityManagerFactory` method allows standard and vendor-specific properties to be passed in and applied to the Entity Manager Factory being created. However, some properties cannot be honored by the aforementioned method. For example, the `javax.persistence.provider` JPA prop-
property, as a means to specify a specific JPA Provider at runtime, cannot be supported because the JPA Provider has already been decided; it is the JPA Provider that registered the Entity Manager Factory Builder service. A JPA Provider should throw an Exception if it recognizes the property but it cannot use the property when specified through the builder. Unrecognized properties must be ignored.

Once an Entity Manager Factory is created the specified Data Source becomes associated with the Entity Manager Factory. It is therefore not possible to re-associate an Entity Manager Factory with another Data Source by providing different properties. A JPA Provider must throw an Exception when an attempt is made to re-specify the database properties. See Rebinding on page 486 for further information.

As an example, a sample snippet of a client that wants to operate on a persistence unit named Accounting and pass in the JDBC user name and password properties is:

```java
ServiceReference[] refs = context.getServiceReferences(
                EntityManagerFactoryBuilder.class.getName(),
                "(osgi.unit.name=Accounting)");
if ( refs != null ) {
    EntityManagerFactoryBuilder emfBuilder =
        (EntityManagerFactoryBuilder) context.getService(refs[0]);
    if ( emfBuilder != null ) {
        Map<String,Object> props = new HashMap<String,Object>();
        props.put("javax.persistence.jdbc.user", userString);
        props.put("javax.persistence.jdbc.password",passwordString);
        EntityManagerFactory emf = emfBuilder.createEntityManagerFactory(props);
        EntityManager em = emf.createEntityManager();
        ...
    }
}
```

The example does not handle the dynamic dependencies on the associated Data Source Factory service.

### 127.4 Extending a Persistence Bundle

A Persistence Bundle is identified by its Meta-Persistence manifest header that references a number of Persistence Descriptor resources. Persistence bundles must be detected by a JPA Provider. The JPA Provider must parse any Persistence Descriptors in these bundles and detect the assigned Persistence Units. For each assigned Persistence Unit, the JPA Provider must register an Entity Manager Factory Builder service when the Persistence Bundle is ready, see Ready Phase on page 483.

For complete and assigned Persistence Units, the JPA Provider must find the required Data Source Factory service based on the driver name. When the Persistence Bundle is ready and the selected DataSource Factory is available, the JPA Provider must have an Entity Manager Factory service registered that is linked to that DataSource Factory.

When the Persistence Bundle is stopped (or the JPA Provider stops), the JPA Provider must close all connections and cleanup any resources associated with the Persistence Bundle.

This process is outlined in detail in the following sections.

#### 127.4.1 Class Space Consistency

A JPA Provider must ignore Persistence Bundles that are in another class space for the javax.persistence.* packages. Such a JPA Provider cannot create JPA Services that would be visible and usable by the Client Bundles.
127.4.2 Meta Persistence Header

A Persistence Bundle is a bundle that contains the Meta-Persistence header. If this header is not present, then this specification does not apply and a JPA Provider should ignore the corresponding bundle.

The persistence root of a Persistence Unit is the root of the Persistence Bundle's JAR.

The Meta-Persistence header has a syntax of:

```
Meta-Persistence ::= ( jar-path ( ',' jar-path)* )?
jar-path        ::= path ( '!' spath )?
spath           ::= path   // must not start with solidus ('/' \u002F)
```

The header may include zero or more comma-separated jar-paths, each a path to a Persistence Descriptor resource in the bundle. Paths may optionally be prefixed with the solidus ('/' \u002F) character. The JPA Provider must always include the META-INF/persistence.xml first if it is not one of the listed paths. Wildcards in directories are not supported. The META-INF/persistence.xml is therefore the default location for an empty header.

For example:

```
Meta-Persistence: META-INF/jpa.xml, persistence/jpa.xml
```

The previous example will instruct the JPA Provider to process the META-INF/persistence.xml resource first, even though it is not explicitly listed. The JPA Provider must then subsequently process META-INF/jpa.xml and the persistence/jpa.xml resources.

The paths in the Meta-Persistence header must be used with the Bundle.getEntry() method, or a mechanism with similar semantics, to obtain the corresponding resource. The getEntry method does not force the bundle to resolve when still unresolved; resolving might interfere with the efficiency of any required entity class enhancements. However, the use of the getEntry method implies that fragment bundles cannot be used to contain Persistence Descriptors nor entity classes.

Paths in the Meta-Persistence header can reference JAR files that are nested in the bundle by using the!/ jar: URL syntax to separate the JAR file from the path within the JAR, for example:

```
Meta-Persistence: embedded.jar!/META-INF/persistence.xml
```

This example refers to a resource in the embedded.jar resource, located in the META-INF directory of embedded.jar.

The!/ splits the jar-path in a prefix and a suffix:

- **Prefix** - The prefix is a path to a JAR resource in the bundle.
- **Suffix** - The suffix is a path to a resource in the JAR identified by the prefix.

For example:

```
embedded.jar!/META-INF/persistence.xml
prefix:   embedded.jar
suffix:    META-INF/persistence.xml
```

It is not required that all listed or implied resources are present in the bundle's JAR. For example, it is valid that the default META-INF/persistence.xml resource is absent. However, if no Persistence Units are found at all then the absence of any Persistence Unit is regarded as an error that should be logged. In this case, the Persistence Bundle is further ignored.

127.4.3 Processing

A JPA Provider can detect a Persistence Bundle as early as its installation time. This early detection allows the JPA Provider to validate the Persistence Bundle as well as prepare any mechanisms to en-
hance the classes for better performance. However, this process can also be delayed until the bundle is started.

The JPA Provider must validate the Persistence Bundle. A valid Persistence Bundle must:

- Have no parsing errors of the Persistence Descriptors
- Validate all Persistence Descriptors against their schemas
- Have at least one assigned Persistence Unit
- Have all entity classes mentioned in the assigned Persistence Units on the Persistence Bundle's JAR.

A Persistence Bundle that uses multiple providers for its Persistence Units could become incompatible with future versions of this specification.

If any validation fails, then this is an error and should be logged. Such a bundle is ignored completely even if it also contains valid assigned Persistence Units. Only a bundle update can recover from this state.

Persistence Units can restrict JPA Providers by specifying a provider dependency. JPA Providers that do not own this JPA Provider implementation class must ignore such a Persistence Unit completely. Otherwise, if the JPA Provider can service a Persistence Unit, it assigns itself to this Persistence Unit.

If after the processing of all Persistence Descriptors, the JPA Provider has no assigned Persistence Units, then the JPA Provider must further ignore the Persistence Bundle.

127.4.4 Ready Phase

A Persistence Bundle is ready when its state is ACTIVE or, when a lazy activation policy is used, STARTING. A JPA Provider must track the ready state of Persistence Bundles that contain assigned Persistence Units.

While a Persistence Bundle is ready, the JPA Provider must have, for each assigned Persistence Unit, an Entity Manager Factory Builder service registered to allow Client Bundles to create new Entity-ManagerFactory objects. The JPA Provider must also register an Entity Manager Factory for each assigned and complete Persistence Unit that has its corresponding Data Source available in the service registry.

The service registration process is asynchronous with the Persistence Bundle start because a JPA Provider could start after a Persistence Bundle became ready.

127.4.5 Service Registrations

The JPA Services must be registered through the Bundle Context of the corresponding Persistence Bundle to ensure proper class space consistency checks by the OSGi Framework.

JPA Services are always related to an assigned Persistence Unit. To identify this Persistence Unit and the assigned JPA Provider, each JPA Service must have the following service properties:

- osgi.unit.name - (String) The name of the Persistence Unit. This property corresponds to the name attribute of the persistence-unit element in the Persistence Descriptor. It is used by Client Bundles as the primary filter criterion to obtain a JPA Service for a required Persistence Unit.
  There can be multiple JPA Services registered under the same osgi.unit.name, each representing a different version of the Persistence Unit.
- osgi.unit.version - (String) The version of the Persistence Bundle, as specified in Bundle-Version header, that provides the corresponding Persistence Unit. Client Bundles can filter their required JPA Services based on a particular Persistence Unit version.
- osgi.unit.provider - (String) The JPA Provider implementation class name that registered the service. The osgi.unit.provider property allows Client Bundles to know the JPA Provider that is servicing the Persistence Unit. Client Bundles should be careful when filtering on this proper-
ty, however, since the JPA Provider that is assigned a Persistence Unit may not be known by the
Client Bundle ahead of time. If there is a JPA Provider dependency, it is better to specify this de-
pendency in the Persistence Unit because other JPA Providers are then not allowed to assign such
a Persistence Unit and will therefore not register a service.

### 127.4.6 Registering the Entity Manager Factory Builder Service

Once the Persistence Bundle is ready, a JPA Provider must register an Entity Manager Factory Builder
service for each assigned Persistence Unit from that Persistence Bundle.

The Entity Manager Factory Builder service must be registered with the service properties listed in
Service Registrations on page 483. The Entity Manager Factory Builder service is registered under
the org.osgi.service.jpa.EntityManagerFactoryBuilder name. This interface is using the JPA pack-
ages and is therefore bound to one of the two supported versions, see Dependencies on page 474.

The Entity Manager Factory Builder service enables the creation of a parameterized version of an
Entity Factory Manager by allowing the caller to specify configuration properties. This approach is
necessary if, for example, the Persistence Unit is not complete.

### 127.4.7 Registering the Entity Manager Factory

A complete Persistence Unit is configured with a specific relational database driver, see JDBC Ac-
cess in JPA on page 478. A JPA Provider must have an Entity Manager Factory service registered for
each assigned and complete Persistence Unit when:

- The originating Persistence Bundle is ready, and
- A matching Data Source Factory service is available. Matching a Data Source Factory service to a
  Persistence Unit is discussed in Database Access on page 485.

A JPA Provider must track the life cycle of the matching Data Source Factory service; while this ser-
vice is unavailable the Entity Manager Factory service must also be unavailable. Any active Entity
Managers created by the Entity Manager Factory service become invalid to use at that time.

The Entity Manager Factory service must be registered with the same service properties as described
for the Entity Manager Factory Builder service, see Service Registrations on page 483. It should be
registered under the following name:

```
javax.persistence.EntityManagerFactory
```

The EntityManagerFactory interface is from the JPA packages and is therefore bound to one of the
two supported versions, see Dependencies on page 474.

An Entity Manager Factory is bound to a Data Source Factory service because its assigned Persis-
tence Unit was complete. However, a Client Bundle could still provide JDBC configuration prop-
erties for the createEntityManager(Map) method. This not always possible, see Rebinding on page
486.

### 127.4.8 Stopping

If a Persistence Bundle is being stopped, then the JPA Provider must ensure that any resources allo-
cated on behalf of the Persistence Bundle are cleaned up and all open connections are closed. This
cleanup must happen synchronously with the STOPPING event. Any Exceptions being thrown
while cleaning up should be logged but must not stop any further clean up.

If the JPA Provider is being stopped, the JPA Provider must unregister all JPA Services that it regis-
tered through the Persistence Bundles and clean up as if those bundles were stopped.
127.5 **JPA Provider**

JPA Providers supply the implementation of the JPA Services and the Persistence Provider service. It is the responsibility of a JPA Provider to store and retrieve the entity classes from a relational database. It is the responsibility of the JPA Provider to register a Persistence Provider and start tracking Persistence Bundles, see *Extending a Persistence Bundle* on page 481.

### 127.5.1 Managed Model

A JPA Provider that supports running in managed mode should register a specific service for the Java EE Containers: the Persistence Provider service. The interface is the standard JPA Persistence-Provider interface. *See Dependencies* on page 474 for the issues around the multiple versions that this specification supports.

The service must be registered with the following service property:

- `javax.persistence.provider` - The JPA Provider implementation class name, a documented name for all JPA Providers.

The Persistence Provider service enables a Java EE Container to find a particular JPA Provider. This service is intended for containers only, not for Client Bundles because there are implicit assumptions in the JPA Providers about the Java EE environment. A Java EE Container must obey the life cycle of the Persistence Provider service. If this service is unregistered then it must close all connections and clean up the corresponding resources.

### 127.5.2 Database Access

A Persistence Unit is configured to work with a relational database. JPA Providers must communicate with a relational database through a compliant JDBC database driver. The database and driver parameters are specified with properties in the Persistence Unit or the configuration properties when a Entity Manager Factory Builder is used to build an Entity Manager Factory. All JPA Providers, regardless of version, in an OSGi environment must support the following properties for database access:

- `javax.persistence.jdbc.driver` - Fully-qualified name of the driver class.
- `javax.persistence.jdbc.url` - Driver-specific URL to indicate database information
- `javax.persistence.jdbc.user` - User name to use when obtaining connections
- `javax.persistence.jdbc.password` - Password to use when obtaining connections

There are severe limitations in specifying these properties after the Entity Manager Factory is created for the first time, see *Rebinding* on page 486.

### 127.5.3 Data Source Factory Service Matching

Providers must use the `javax.persistence.jdbc.driver` property, as defined in *JDBC Access in JPA* on page 478, to obtain a Data Source Factory service. The Data Source Factory is specified in *JDBC™ Service Specification* on page 445. The `javax.persistence.jdbc.driver` property must be matched with the value of the Data Source Factory service property named `osgi.jdbc.driver.class`.

The Data Source Factory service is registered with the `osgi.jdbc.driver.class` service property that holds the class name of the driver. This property must match the `javax.persistence.jdbc.driver` service property of the Persistence Unit.

For example, if the Persistence Unit specifies the `com.acme.db.Driver` database driver in the `javax.persistence.jdbc.driver` property (or in the Persistence Descriptor property element), then the following filter would select an appropriate Data Source Factory:

- `osgi.jdbc.driver.class=com.acme.db.Driver`
Once the Data Source Factory is obtained, the JPA Provider must obtain a DataSource object. This DataSource object must then be used for all relational database access.

In [1] JPA 1.0 the JPA JDBC properties were not standardized. JPA Providers typically defined a set of JDBC properties, similar to those defined in JPA 2.0, to configure JDBC driver access. JPA 1.0 JPA Providers must look up the Data Source Factory service first using the JPA 2.0 JDBC properties. If these properties are not defined then they should fall back to their proprietary driver properties.

127.5.4 Rebinding

In this specification, the Entity Manager Factory service is only registered when the Persistence Unit is complete and a matching Data Source Factory service is available. However, the API of the Entity Manager Factory Builder allows the creation of an Entity Manager Factory with configuration properties. Those configuration properties could contain the JDBC properties to bind to another Data Source Factory service than it had already selected.

This case must not be supported by a JPA Provider, an Illegal Argument Exception must be thrown. If such a case would be supported then the life cycle of the Entity Manager Factory service would still be bound to the first Data Source Factory. There would be no way for the JPA Provider to signal to the Client Bundle that the returned Entity Manager Factory is no longer valid because the rebound Data Source Factory was unregistered.

Therefore, when an Entity Manager Factory is being created using the Entity Manager Factory Builder, a JPA Provider must verify that the new properties are compatible with the properties of the already created Entity Manager Factory. If no, then an Exception must be thrown. If they are compatible, then an instance of the previous Entity Manager Factory should be returned.

127.5.5 Enhancing Entity Classes

JPA Providers may choose to implement the JPA specifications using various implementation approaches and techniques. This promotes innovation in the area, but also opens the door to limitations and constraints arising due to implementation choices. For example, there are JPA Providers that perform byte code weaving during the entity class loading. Dynamic byte code weaving requires that the entity classes are not loaded until the JPA Provider is first able to intercept the loading of the entity class and be given an opportunity to do its weaving. It also implies that the Persistence Bundle and any other bundles that import packages from that bundle must be refreshed if the JPA Provider needs to be changed.

This is necessary because the JPA Services are registered against the Bundle Contexts of the Persistence Bundles and not the Bundle Context of the JPA Providers. Client Bundles must then unget the service to unbind themselves from the uninstalled JPA Provider. However, since most JPA Providers perform some kind of weaving or class transformation on the entity classes, the Persistence Bundle will likely need to be refreshed. This will cause the Client Bundles to be refreshed also because they depend on the packages of the entity classes.

127.5.6 Class Loading

JPA Providers cannot have package dependencies on entity classes in Persistence Bundles because they cannot know at install time what Persistence Bundles they will be servicing. However, when a JPA Provider is servicing a Persistence Bundle, it must be able to load classes and resources from that Persistence Bundle according to the OSGi bundle rules. To do this class loading it must obtain a class loader that has the same visibility as the Persistence Bundle's bundle class loader. This will also allow it to load and manage metadata for the entity classes and resources for that Persistence Bundle's assigned Persistence Units. These resources and entity classes must reside directly in the Persistence Bundle, they must be accessed using the getEntry method. Entity classes and resources must not reside in fragments.
**Validation**

There is not yet an OSGi service specification defined for validation providers. If validation is required, the validation implementation will need to be included with the JPA Provider bundle.

**Static Access**

Non-managed client usage of JPA has traditionally been achieved through the `Persistence` class. Invoking a static method on the `Persistence` class is a dependency on the returned JPA Provider that cannot be managed by the OSGi framework.

However, such an unmanaged dependency is supported in this specification by the Static Persistence bundle. This bundle provides backwards compatibility for programs that use existing JPA access patterns. However, usage of this static model requires that the deployer ensures that the actors needed are in place at the appropriate times by controlling the life cycles of all participating bundles. The normal OSGi safe-guards and dependency handling do not work in the case of static access.

A Static Persistence Bundle must provide static access from the `Persistence` class to the JPA Services.

**Access**

There are two methods on the `Persistence` class:

- `createEntityManagerFactory(String)`
- `createEntityManagerFactory(String, Map)`

Both methods take the name of a Persistence Unit. The last method also takes a map that contains extra configuration properties. To support the usage of the static methods on the `Persistence` class, the implementation of the `Persistence.createEntityManagerFactory` method family must do a lookup of one of the JPA Services associated with the selected Persistence Unit.

If no configuration properties are specified, the Static Persistence Bundle must look for an Entity Manager Factory service with the `osgi.unit.name` property set to the given name. The default service should be used because no selector for a version is provided. If no such service is available, `null` must be returned. Provisioning of multiple versioned Persistence Units is not supported. Deployers should ensure only a single version of a Persistence Unit with the same name is present in an OSGi framework at any moment in time.

Otherwise, if configuration properties are provided, the Static Access implementation must look for an Entity Manager Factory Builder service with the `osgi.unit.name` property set to the given Persistence Unit name. If no such service exists, `null` must be returned. Otherwise, the default service must be used to create an Entity Manager Factory with the given configuration properties. The result must be returned to the caller.

For service lookups, the Static Persistence Bundle must use its own Bundle Context, it must not attempt to use the Bundle Context of the caller. All exceptions should be passed to the caller.

The class space of the Entity Manager Factory and the class space of the client cannot be enforced to be consistent by the framework because it is the `Persistence` class that is doing the lookup of the service, and not the actual calling Client Bundle that will be using the Entity Manager Factory. The framework cannot make the connection and therefore cannot enforce that the class spaces correspond. Deployers should therefore ensure that the involved class spaces are correctly wired.

**Security**

The security for this specification is based on the JPA specification.
127.8 org.osgi.service.jpa

JPA 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.jpa; version="[1.0,2.0)"

Example import for providers implementing the API in this package:
import-package: org.osgi.service.jpa; version="[1.0,1.1)"

127.8.1 public interface EntityManagerFactoryBuilder

This service interface offers JPA clients the ability to create instances of EntityManagerFactory for a given named persistence unit. A service instance will be created for each named persistence unit and can be filtered by comparing the value of the osgi.unit.name property containing the persistence unit name. This service is used specifically when the caller wants to pass in factory-scoped properties as arguments. If no properties are being used in the creation of the EntityManagerFactory then the basic EntityManagerFactory service should be used.

127.8.1.1 public static final String JPA_UNIT_NAME = "osgi.unit.name"
The name of the persistence unit.

127.8.1.2 public static final String JPA_UNIT_PROVIDER = "osgi.unit.provider"
The class name of the provider that registered the service and implements the JPA javax.persistence.PersistenceProvider interface.

127.8.1.3 public static final String JPA_UNIT_VERSION = "osgi.unit.version"
The version of the persistence unit bundle.

127.8.1.4 public EntityManagerFactory createEntityManagerFactory(Map<String,Object> props)

props Properties to be used, in addition to those in the persistence descriptor, for configuring the EntityManagerFactory for the persistence unit.

□ Return an EntityManagerFactory instance configured according to the properties defined in the corresponding persistence descriptor, as well as the properties passed into the method.

Returns An EntityManagerFactory for the persistence unit associated with this service. Must not be null.

127.9 References

[1] JPA 1.0

[2] JPA 2.0

[3] Java EE 5
128 Web Applications Specification

Version 1.0

128.1 Introduction

The Java EE Servlet model has provided the backbone of web based applications written in Java. Given the popularity of the Servlet model, it is desirable to provide a seamless experience for deploying existing and new web applications to Servlet containers operating on the OSGi framework. Previously, the Http Service in the catalogue of OSGi compendium services was the only model specified in OSGi to support the Servlet programming model. However, the Http Service, as defined in that specification, is focused on the run time, as well as manual construction of the servlet context, and thus does not actually support the standard Servlet packaging and deployment model based on the Web Application Archive, or WAR format.

This specification defines the Web Application Bundle, which is a bundle that performs the same role as the WAR in Java EE. A WAB uses the OSGi life cycle and class/resource loading rules instead of the standard Java EE environment. WABs are normal bundles and can leverage the full set of features of the OSGi framework.

Web applications can also be installed as traditional WARs through a manifest rewriting process. During the install, a WAR is transformed into a WAB. This specification was based on ideas developed in [5] PAX Web Extender.

This Web Application Specification provides support for web applications written to the Servlet 2.5 specification, or later. Given that Java Server Pages, or JSPs, are an integral part of the Java EE web application framework, this specification also supports the JSP 2.1 specification or greater if present.

This specification details how a web application packaged as a WAR may be installed into an OSGi framework, as well as how this application may interact with, and obtain, OSGi services.

128.1.1 Essentials

- **Extender** - Enable the configuration of components inside a bundle based on configuration data provided by the bundle developer.
- **Services** - Enable the use of OSGi services within a Web Application.
- **Deployment** - Define a mechanism to deploy Web Applications, both OSGi aware and non OSGi aware, in the OSGi environment.
- **WAR File Support** - Transparently enhance the contents of a WAR’s manifest during installation to add any headers necessary to deploy a WAR as an OSGi bundle.

128.1.2 Entities

- **Web Container** - The implementation of this specification. Consists of a Web Extender, a Web URL Handler and a Servlet and Java Server Pages Web Runtime environment.
- **Web Application Archive (WAR)** - The Java EE standard resource format layout of a JAR file that contains a deployable Web Application.
- **Web Application Bundle** - A Web Application deployed as an OSGi bundle, also called a WAB.
- **WAB** - The acronym for a Web Application Bundle.
- **Web Extender** - An extender bundle that deploys the Web Application Bundle to the Web Runtime based on the Web Application Bundle’s state.

- **Web URL Handler** - A URL handler which transforms a Web Application Archive (WAR) to conform to the OSGi specifications during installation by installing the WAR through a special URL so that it becomes a Web Application Bundle.

- **Web Runtime** - A Java Server Pages and Servlet environment, receiving the web requests and translating them to servlet calls, either from Web Application servlets or other classes.

- **Web Component** - A Servlet or Java Server Page (JSP).

- **Servlet** - An object implementing the Servlet interface; this is for the request handler model in the Servlet Specification.

- **Servlet Context** - The model representing the Web Application in the Servlet Specification.

- **Java Server Page (JSP)** - A declarative, template based model for generating content through Servlets that is optionally supported by the Web Runtime.

- **Context Path** - The URI path prefix of any content accessible in a Web Application.

---

**Figure 128.1 Web Container Entities**

![Diagram of Web Container Entities]

---

### 128.1.3 Dependencies

The package dependencies for the clients of this specification are listed in the following table.

<table>
<thead>
<tr>
<th>Packages</th>
<th>Export Version</th>
<th>Client Import Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.servlet</td>
<td>2.5</td>
<td>[2.5,3.0)</td>
</tr>
<tr>
<td>javax.servlet.http</td>
<td>2.5</td>
<td>[2.5,3.0)</td>
</tr>
<tr>
<td>javax.servlet.jsp.el</td>
<td>2.1</td>
<td>[2.1,3.0)</td>
</tr>
<tr>
<td>javax.servlet.jsp.jstl.core</td>
<td>1.2</td>
<td>[1.2,2.0)</td>
</tr>
<tr>
<td>javax.servlet.jsp.jstl.fmt</td>
<td>1.2</td>
<td>[1.2,2.0)</td>
</tr>
<tr>
<td>javax.servlet.jsp.jstl.sql</td>
<td>1.2</td>
<td>[1.2,2.0)</td>
</tr>
<tr>
<td>javax.servlet.jsp.jstl.tlv</td>
<td>1.2</td>
<td>[1.2,2.0)</td>
</tr>
<tr>
<td>javax.servlet.jsp.resources</td>
<td>2.1</td>
<td>[2.1,3.0)</td>
</tr>
</tbody>
</table>
JSP is optional for the Web Runtime.

128.1.4 Synopsis

The Web Application Specification is composed of a number of cooperating parts, which are implemented by a Web Container. A Web Container consists of:

- **Web Extender** - Responsible for deploying Web Application Bundles (WAB) to a Web Runtime,
- **Web Runtime** - A runtime environment for a Web Application that supports Servlet 2.5 specification and JSP 2.1 specification or later. The Web Runtime receives web requests and calls the appropriate methods on servlets. Servlets can be implemented by classes or Java Server Pages.
- **Web URL Handler** - A URL stream handler providing the webbundle: scheme. This scheme can be used to install WARs in an OSGi framework. The Web URL Handler will then automatically add the required OSGi manifest headers.

WABs are standard OSGi bundles with additional headers in the manifest that serve as deployment instructions to the Web Extender. WABs can also contain the Java EE defined web.xml descriptor in the WEB-INF/ directory. When the Web Extender detects that a WAB is ready, the Web Extender deploys the WAB to the Web Runtime using information contained in the web.xml descriptor and the appropriate manifest headers. The Bundle Context of the WAB is made available as a Servlet Context attribute. From that point, the Web Runtime will use the information in the WAB to serve content to any requests. Both dynamic as well as static content can be provided.

The Web URL Handler allows the deployment of an unmodified WAR as a WAB into the OSGi framework. This Web URL Handler provides a URL stream handler with the webbundle: scheme. Installing a WAR with this scheme allows the Web URL Handler to interpose itself as a filter on the input stream of the contents of the WAR, transforming the contents of the WAR into a WAB. The Web URL Handler rewrites the manifest by adding necessary headers to turn the WAR into a valid WAB. Additional headers can be added to the manifest that serve as instructions to the Web Extender.

After a WAB has been deployed to the Web Runtime, the Web Application can interact with the OSGi framework via the provided Bundle Context. The Servlet Context associated with this WAB follows the same life cycle as the WAB. That is, when the underlying Web Application Bundle is started, the Web Application is deployed to the Web Runtime. When the underlying Web Application Bundle is stopped because of a failure or other reason, the Web Application is undeployed from the Web Runtime.

128.2 Web Container

A Web Container is the implementation of this specification. It consists of the following parts:

- **Web Extender** - Detects Web Application Bundles (WAB) and tracks their life cycle. Ready WABs are deployed to the Web Runtime.
- **Web Runtime** - A runtime environment for a Web Application that supports the Servlet 2.5 specification and JSP 2.1 specification or later. The Web Runtime receives web requests and calls the appropriate methods on servlets. Servlets can be implemented by classes or Java Server Pages.
- **Web URL Handler** - A URL stream handler providing the webbundle: scheme. This scheme can be used to install WARs in an OSGi framework. The Web URL Handler will then automatically add the required OSGi manifest headers.

The extender, runtime, and handler can all be implemented in the same or different bundles and use unspecified mechanisms to communicate. This specification uses the defined names of the subparts as the actor; the term Web Container is the general name for this collection of actors.
128.3 **Web Application Bundle**

Bundles are the deployment and management entities under OSGi. A **Web Application Bundle** (WAB) is deployed as an OSGi bundle in an OSGi framework, where each WAB provides a single **Web Application**. A Web Application can make use of the [3] Servlet 2.5 specification and [4] JSP 2.1 specification programming models, or later, to provide content for the web.

A WAB is defined as a normal OSGi bundle that contains web accessible content, both static and dynamic. There are no restrictions on bundles. A Web Application can be packaged as a WAB during application development, or it can be transparently created at bundle install time from a standard Web Application Archive (WAR) via transformation by the Web URL Handler, see **Web URL Handler** on page 496.

A WAB is a valid OSGi bundle and as such must fully describe its dependencies and exports (if any). As Web Applications are modularized further into multiple bundles (and not deployed as WAR files only) it is possible that a WAB can have dependencies on other bundles.

A WAB may be installed into the framework using the `BundleContext.installBundle` methods. Once installed, a WAB's life cycle is managed just like any other bundle in the framework. This life cycle is tracked by the Web Extender who will then deploy the Web Application to the Web Runtime when the WAB is ready and will undeploy it when the WAB is no longer ready. This state is depicted in Figure 128.2.

**Figure 128.2 State diagram Web Application**

```
init
→ collision resolved
WAB started

DEPLOYING

DEPLOYED
→ Web Application deployed to runtime
WAB or Web Extender stopped

FAILED
→ failure

UNDEPLOYED
→ Web Application no longer available

UNDEPLOYING
```

### 128.3.1 WAB Definition

A WAB is differentiated from non Web Application bundles through the specification of the additional manifest header:

```
Web-ContextPath ::= path
```

The Web-ContextPath header specifies the value of the **Context Path** of the Web Application. All web accessible content of the Web Application is available on the web server relative to this Context Path. For example, if the context path is /sales, then the URL would be something like: `http://www.acme.com/sales`. The Context Path must always begin with a solidus (`"/"`) or zero or more forward slashes (`"/"`).

The Web Extender must not recognize a bundle as a Web Application unless the Web-ContextPath header is present in its manifest and the header value is a valid path for the bundle.

A WAB can optionally contain a `web.xml` resource to specify additional configuration. This `web.xml` must be found with the Bundle `findEntries` method at the path:
WEB-INF/web.xml

The findEntries method includes fragments, allowing the web.xml to be provided by a fragment. The Web Extender must fully support a web.xml descriptor that specifies Servlets, Filters, or Listeners whose classes are required by the WAB.

128.3.2 Starting the Web Application Bundle

A WAB's Web Application must be deployed while the WAB is ready. Deployed means that the Web Application is available for web requests. Once deployed, a WAB can serve its web content on the given Context Path. Ready is when the WAB:

- Is in the ACTIVE state, or
- Has a lazy activation policy and is in the STARTING state.

The Web Extender should ensure that serving static content from the WAB does not activate the WAB when it has a lazy activation policy.

To deploy the WAB, the Web Extender must initiate the deploying of the Web Application into a Web Runtime. This is outlined in the following steps:

1. Wait for the WAB to become ready. The following steps can take place asynchronously with the starting of the WAB.
2. Post an org/osgi/service/web/DEPLOYING event. See Events on page 499.
3. Validate that the Web-ContextPath manifest header does not match the Context Path of any other currently deployed web application. If the Context Path value is already in use by another Web Application, then the Web Application must not be deployed, and the deployment fails, see Failure on page 494. The Web Extender should log the collision. If the prior Web Application with the same Context Path is undeployed later, this Web Application should be considered as a candidate, see Stopping the Web Application Bundle on page 495.
4. The Web Runtime processes deployment information by processing the web.xml descriptor, if present. The Web Container must perform the necessary initialization of Web Components in the WAB as described in the [3] Servlet 2.5 specification. This involves the following sub-steps in the given order:
   - Create a Servlet Context for the Web Application.
   - Instantiate configured Servlet event listeners.
   - Instantiate configured application filter instances etc.

   The Web Runtime is required to complete instantiation of listeners prior to the start of execution of the first request into the Web Application by the Web Runtime. Attribute changes to the Servlet Context and Http Session objects can occur concurrently. The Servlet Container is not required to synchronize the resulting notifications to attribute listener classes. Listener classes that maintain state are responsible for the integrity of the data and should handle this case explicitly.

   If event listeners or filters are used in the web.xml, then the Web Runtime will load the corresponding classes from the bundle activating the bundle if it was lazily started. Such a configuration will therefore not act lazily.
5. Publish the Servlet Context as a service with identifying service properties, see Publishing the Servlet Context on page 494.
6. Post an org/osgi/service/web/DEPLOYED event to indicate that the web application is now available. See Events on page 499.

If at any moment before the org/osgi/service/web/DEPLOYED event is published the deployment of the WAB fails, then the WAB deployment fails, see Failure on page 494.
128.3.3 Failure

Any validation failures must prevent the Web Application from being accessible via HTTP, and must result in a `org/osgi/service/web/FAILED` event being posted. See Events on page 499. The situation after the failure must be as if the WAB was never deployed.

128.3.4 Publishing the Servlet Context

To help management agents with tracking the state of Web Applications, the Web Extender must register the Servlet Context of the WAB as a service, using the Bundle Context of the WAB. The Servlet Context service must be registered with the service properties listed in the following table.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>osgi.web.symbolicname</td>
<td>String</td>
<td>The symbolic name for the Web Application Bundle</td>
</tr>
<tr>
<td>osgi.web.version</td>
<td>String</td>
<td>The version of the Web Application Bundle. If no Bundle-Version is specified in the manifest then this property must not be set.</td>
</tr>
<tr>
<td>osgi.web.contextpath</td>
<td>String</td>
<td>The Context Path from which the WAB's content will be served.</td>
</tr>
</tbody>
</table>

128.3.5 Static Content

A deployed WAB provides content on requests from the web. For certain access paths, this can serve content from the resources of the web application: this is called static content. A Web Runtime must use the Servlet Context resource access methods to service static content, the resource loading strategy for these methods is based on the `findEntries` method, see Resource Lookup on page 500. For confidentiality reasons, a Web Runtime must not return any static content for paths that start with one of the following prefixes:

- `WEB-INF/`
- `OSGI-INF/`
- `META-INF/`
- `OSGI-OPT/`

These protected directories are intended to shield code content used for dynamic content generation from accidentally being served over the web, which is a potential attack route. In the servlet specification, the WEB-INF/ directory in the WAR is protected in such a way. However, this protection is not complete. A dependent JAR can actually be placed outside the WEB-INF directory that can then be served as static content. The same is true for a WAB. Though the protected directories must never be served over the web, there are no other checks required to verify that no content can be served that is also available from the Bundle class path.

It is the responsibility of the author of the WAB to ensure that confidential information remains confidential by placing it in one of the protected directories. WAB bundles should be constructed in such a way that they do not accidentally expose code or confidential information. The simplest way to achieve this is to follow the WAR model where code is placed in the WEB-INF/classes directory and this directory is placed on the Bundle's class path as the first entry. For example:

```
Bundle-ClassPath: WEB-INF/classes, WEB-INF/lib/acme.jar
```

128.3.6 Dynamic Content

Dynamic content is content that uses code to generate the content, for example a servlet. This code must be loaded from the bundle with the Bundle `loadClass` method, following all the Bundle class path rules.
Unlike a WAR, a WAB is not constrained to package classes and code resources in the WEB-INF/classes directory or dependent JARs in WEB-INF/lib/ only. These entries can be packaged in any way that's valid for an OSGi bundle as long as such directories and JARs are part of bundle class path as set with the Bundle-ClassPath header and any attached fragments. JARs that are specified in the Bundle-ClassPath header are treated like JARs in the WEB-INF/lib/ directory of the Servlet specification. Similarly, any directory that is part of the Bundle-ClassPath header is treated like WEB-INF/classes directory of the Servlet specification.

Like WARs, code content that is placed outside the protected directories can be served up to clients as static content.

### 128.3.7 Content Serving Example

This example consists of a WAB with the following contents:

```plaintext
acme.jar:
  Bundle-ClassPath: WEB-INF/classes, LIB/bar.jar
  Web-ContextPath: /acme

WEB-INF/lib/foo.jar
LIB/bar.jar
index.html
favicon.ico
```

The content of the embedded JARs `foo.jar` and `bar.jar` is:

```plaintext
foo.jar:
  META-INF/foo.tld
  foo/FooTag.class

bar.jar:
  META-INF/bar.tld
  bar/BarTag.class
```

Assuming there are no special rules in place then the following lists specify the result of a number of web requests for static content:

- `/acme/index.html`:
  - `acme.wab:index.html`
- `/acme/favicon.ico`:
  - `acme.wab:favicon.ico`
- `/acme/WEB-INF/lib/foo.jar`:
  - not found because protected directory
- `/acme/LIB/bar.jar`:
  - `acme.wab:LIB/bar.jar` (code, but not protected)

In this example, the tag classes in `bar.jar` must be found (if JSP is supported) but the tag classes in `foo.jar` must not because `foo.jar` is not part of the bundle class path.

### 128.3.8 Stopping the Web Application Bundle

A web application is stopped by stopping the corresponding WAB. In response to a WAB STOPPING event, the Web Extender must undeploy the corresponding Web Application from the Servlet Container and clean up any resources. This undeploying must occur synchronously with the WAB's stopping event. This will involve the following steps:

1. An `org/osgi/service/web/UNDEPLOYING` event is posted to signal that a Web Application will be removed. See Events on page 499.
2. Unregister the corresponding Servlet Context service
3. The Web Runtime must stop serving content from the Web Application.
4. The Web Runtime must clean up any Web Application specific resources as per servlet 2.5 specification.
5. Emit an `org/osgi/service/web/UNDEPLOYED` event. See Events on page 499.
6. It is possible that there are one or more colliding WABs because they had the same Context Path as this stopped WAB. If such colliding WABs exist then the Web Extender must attempt to deploy the colliding WAB with the lowest bundle id.
Any failure during undeploying should be logged but must not stop the cleaning up of resources and notification of (other) listeners as well as handling any collisions.

### 128.3.9 Uninstalling the Web Application Bundle

A web application can be uninstalled by uninstalling the corresponding WAB. The WAB will be uninstalled from the OSGi framework.

### 128.3.10 Stopping of the Web Extender

When the Web Extender is stopped all deployed WABs are undeployed as described in Stopping the Web Application Bundle on page 495. Although the WAB is undeployed it remains in the ACTIVE state. When the Web Extender leaves the STOPPING state all WABs will have been undeployed.

### 128.4 Web URL Handler

The Web URL Handler acts as a filter on the Input Stream of an install operation. It receives the WAB or WAR and it then generates a JAR that conforms to the WAB specification by rewriting the manifest resource. This process is depicted in Figure 128.3.

![Figure 128.3 Web URL Handler](image)

When the Web Container bundle is installed it must provide the `webbundle:scheme` to the URL class. The Web URL Handler has two primary responsibilities:

- **WAB** - If the source is already a bundle then only the Web-ContextPath can be set or overwritten.
- **WAR** - If the source is a WAR (that is, it must not contain any OSGi defined headers) then convert the WAR into a WAB.

The Web URL Handler can take parameters from the query arguments of the install URL, see URL Parameters on page 497.

The URL handler must validate query parameters, and ensure that the manifest rewriting results in valid OSGi headers. Any validation failures must result in Bundle Exception being thrown and the bundle install must fail.

Once a WAB is generated and installed, its life cycle is managed just like any other bundle in the framework.
128.4.1 **URL Scheme**

The Web URL Handler’s scheme is defined to be:

```plaintext
scheme ::= 'webbundle:' embedded '?' web-params
embedded ::= <embedded URL according to RFC 1738>
web-params ::= ( web-param ( '&' web-param )* )?
web-param ::= <key> '=' <value>
```

The `web-param <key>` and `<value>` as well as the `<embedded url>` must follow [6] Uniform Resource Locators, RFC 1738 for their escaping and character set rules. A Web URL must further follow all the rules of a URL. Whitespaces are not allowed between terms.

An example for a `webbundle:` URL:

```plaintext
```

Any URL scheme understood by the framework can be embedded, such as an `http:` or `file:` URL. Some forms of embedded URL also contain URL query parameters and this must be supported. The embedded URL must be encoded as a standard URL. That is, the control characters like colon (`:` :), solidus (`/` /), percent (`%` %), and ampersand (`&` &) must not be encoded. Thus the value returned from the `getPath` method may contain a query part. Any implementation must take care to preserve both the query parameters for the embedded URL, and for the complete `webbundle:` URL. A question mark must always follow the embedded URL to simplify this processing. The following example shows an HTTP URL with some query parameters:

```plaintext
```

128.4.2 **URL Parsing**

The URL object for a `webbundle:` URL must return the following values for the given methods:

- `getProtocol` - `webbundle`
- `getPath` - The complete embedded URL
- `getQuery` - The parameters for processing of the manifest.

For the following example:

```plaintext
```

The aforementioned methods must return:

- `getProtocol` - `webbundle`
- `getPath` - `http://acme.com/repo?war=example.war`
- `getQuery` - `Web-ContextPath=/sales`

128.4.3 **URL Parameters**

All the parameters in the `webbundle:` URL are optional except for the Web-ContextPath parameter. The parameter names are case insensitive, but their values must be treated as case sensitive. Table 128.3 describes the parameters that must be supported by any `webbundle:` URL Stream handler. A Web URL Handler is allowed to support additional parameters.

**Table 128.3 Web bundle URL Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bundle-SymbolicName</td>
<td>The desired symbolic name for the resulting WAB.</td>
</tr>
</tbody>
</table>
### 128.4.4 WAB Modification

The Web URL Handler can set or modify the Web-ContextPath of a WAB if the input source is already a bundle. It must be considered as a bundle when any of the OSGi defined headers listed in Table 128.3 is present in the bundle.

For WAB Modification, the Web URL Handler must only support the Web-ContextPath parameter and it must not modify any existing headers other than the Web-ContextPath. Any other parameter given must result in a Bundle Exception.

### 128.4.5 WAR Manifest Processing

The Web URL Handler is designed to support the transparent deployment of Java EE Web ARchives (WAR). Such WARs are ignorant of the requirements of the underlying OSGi framework that hosts the Web Runtime. These WARs are not proper OSGi bundles because they do not contain the necessary metadata in the manifest. For example, a WAR without a Bundle-ManifestVersion, Import-Package, and other headers cannot operate in an OSGi framework.

The Web URL Handler implementation copies the contents of the embedded URL to the output and rewrites the manifest headers based on the given parameters. The result must be a WAB.

Any parameters specified must be treated as manifest headers for the web. The following manifest headers must be set to the following values if not specified:

- **Bundle-ManifestVersion** - Must be set to 2.
- **Bundle-SymbolicName** - Generated in an implementation specific way.
- **Bundle-ClassPath** - Must consist of:
  - WEB-INF/classes
  - All JARs from the WEB-INF/lib directory in the WAR. The order of these embedded JARs is unspecified.
  - If these JARs declare dependencies in their manifest on other JARs in the bundle, then these jars must also be appended to the Bundle-ClassPath header. The process of detecting JAR dependencies must be performed recursively as indicated in the Servlet Specification.
- **Web-ContextPath** - The Web-ContextPath must be specified as a parameter. This Context Path should start with a leading solidus ('/'). The Web URL handler must add the preceding solidus if it is not present.

The Web URL Handler is responsible for managing the import dependencies of the WAR. Implementations are free to handle the import dependencies in an implementation defined way. They can augment the Import-Package header with byte-code analysis information, add a fixed set of clauses, and/or use the DynamicImport-Package header as last resort.

Any other manifest headers defined as a parameter or WAR manifest header not described in this section must be copied to the WAB manifest by the Web URL Handler. Such an header must not be modified.
128.4.6 Signed WAR files

When a signed WAR file is installed using the Web URL Handler, then the manifest rewriting process invalidates the signatures in the bundle. The OSGi specification requires fully signed bundles for security reasons, security resources in partially signed bundles are ignored.

If the use of the signing metadata is required, the WAR must be converted to a WAB during development and then signed. In this case, the Web URL Handler cannot be used. If the Web URL Handler is presented with a signed WAR, the manifest name sections that contain the resource's check sums must be stripped out by the URL stream handler. Any signer files (*.SF and their corresponding DSA/RSA signature files) must also be removed.

128.5 Events

The Web Extender must track all WABs in the OSGi framework in which the Web Extender is installed. The Web Extender must post Event Admin events, which is asynchronous, at crucial points in its processing. The topic of the event must be one of the following values:

- org/osgi/service/web/DEPLOYING - The Web Extender has accepted a WAB and started the process of deploying a Web Application.
- org/osgi/service/web/DEPLOYED - The Web Extender has finished deploying a Web Application, and the Web Application is now available for web requests on its Context Path.
- org/osgi/service/web/UNDEPLOYING - The extender started undeploying the Web Application in response to its corresponding WAB being stopped or the Web Extender is stopped.
- org/osgi/service/web/UNDEPLOYED - The Web Extender has undeployed the Web Application. The application is no longer available for web requests.
- org/osgi/service/web/FAILED - The Web Extender has failed to deploy the Web Application, this event can be fired after the DEPLOYING event has fired and indicates that no DEPLOYED event will be fired.

For each event topic above, the following properties must be published:

- bundle.symbolicName - (String) The bundle symbolic name of the WAB.
- bundle.id - (Long) The bundle id of the WAB.
- bundle - (Bundle) The Bundle object of the WAB.
- bundle.version - (Version) The version of the WAB.
- context.path - (String) The Context Path of the Web Application.
- timestamp - (Long) The time when the event occurred
- extender.bundle - (Bundle) The Bundle object of the Web Extender Bundle
- extender.bundle.id - (Long) The id of the Web Extender Bundle.
- extender.bundle.symbolicName - (String) The symbolic name of the Web Extender Bundle.

In addition, the org/osgi/service/web/FAILED event must also have the following property:

- exception - (Throwable) If an exception caused the failure, an exception detailing the error that occurred during the deployment of the WAB.
- collision - (String) If a name collision occurred, the Web-ContextPath that had a collision
- collision.bundles - (Collection<Long>) If a name collision occurred, a collection of bundle ids that all have the same value for the Web-ContextPath manifest header.
Interacting with the OSGi Environment

128.6 Interacting with the OSGi Environment

128.6.1 Bundle Context Access

In order to properly integrate in an OSGi environment, a Web Application can access the OSGi service registry for publishing its services, accessing services provided by other bundles, and listening to bundle and service events to track the life cycle of these artifacts. This requires access to the Bundle Context of the WAB.

The Web Extender must make the Bundle Context of the corresponding WAB available to the Web Application via the Servlet Context `osgi-bundlecontext` attribute. A Servlet can obtain a Bundle Context as follows:

```java
BundleContext ctxt = (BundleContext) servletContext.getAttribute("osgi-bundlecontext");
```

128.6.2 Other Component Models

Web Applications sometimes need to inter-operate with services provided by other component models, such as a Declarative Services or Blueprint. Per the Servlet specification, the Servlet Container owns the life cycle of a Servlet; the life cycle of the Servlet must be subordinate to the life cycle of the Servlet Context, which is only dependent on the life cycle of the WAB. Interactions between different bundles are facilitated by the OSGi service registry. This interaction can be managed in several ways:

- A Servlet can obtain a Bundle Context from the Servlet Context for performing service registry operations.
- Via the JNDI Specification and the `osgi:service` JNDI namespace. The OSGi JNDI specification describes how OSGi services can be made available via the JNDI URL Context. It defines an `osgi:service` namespace and leverages URL Context factories to facilitate JNDI integration with the OSGi service registry.

Per this specification, it is not possible to make the Servlet life cycle dependent on the availability of specific services. Any synchronization and service dependency management must therefore be done by the Web Application itself.

128.6.3 Resource Lookup

The `getResource` and `getResourceAsStream` methods of the `ServletContext` interface are used to access resources in the web application. For a WAB, these resources must be found according to the `findEntries` method, this method includes fragments. For the `getResource` and `getResourceAsStream` method, if multiple resources are found, then the first one must be used.

Since the `getResource` and `getResourceAsStream` methods do not support wildcards while the `findEntries` method does it is necessary to escape the wildcard asterisk (`*` `\u002A`) with prefixing it with a reverse solidus (`\` `\u005C`). This implies that a reverse solidus must be escaped with an extra reverse solidus. For example, the path `foo\bar*` must be escaped to `foo\\bar\*`.

The `getResourcePaths` method must map to the Bundle `getEntryPaths` method, its return type is a Set and can not handle multiples. However, the paths from the `getEntryPaths` method are relative while the methods of the `getResourcePaths` must be absolute.

For example, assume the following manifest for a bundle:

```manifest
Bundle-ClassPath: localized, WEB-INF
...
```

This WAB has an attached fragment `acme-de.jar` with the following content:
The `getResource` method for `localized/logo.png` uses the `findEntries` method to find a resource in the directory `/localized` and the resource `logo.png`. Assuming the host bundle has no `localized` directory, the Web Runtime must serve the `logo.png` resource from the `acme-de.jar`.

**128.6.4 Resource Injection and Annotations**

The Web Application `web.xml` descriptor can specify the `metadata-complete` attribute on the `web-app` element. This attribute defines whether the `web.xml` descriptor is `complete`, or whether the classes in the bundle should be examined for deployment annotations. If the `metadata-complete` attribute is set to true, the Web Runtime must ignore any servlet annotations present in the class files of the Web Application. Otherwise, if the `metadata-complete` attribute is not specified, or is set to false, the container should process the class files of the Web Application for annotations, if supported.

A WAB can make use of the annotations defined by [7] JSR 250 Common Annotations for the Java Platform if supported by the Web Extender. Such a WAB must import the packages the annotations are contained in. A Web Extender that does not support the use of JSR 250 annotations must not process a WAB that imports the annotations package.

**128.6.5 Java Server Pages Support**

Java Server Pages (JSP) is a rendering technology for template based web page construction. This specification supports [4] JSP 2.1 specification if available with the Web Runtime. The `servlet` element in a `web.xml` descriptor is used to describe both types of Web Components. JSP components are defined implicitly in the `web.xml` descriptor through the use of an implicit `.jsp` extension mapping, or explicitly through the use of a `jsp-group` element.

**128.6.6 Compilation**

A Web Runtime compiles a JSP page into a Servlet, either during the deployment phase, or at the time of request processing, and dispatches the request to an instance of such a dynamically created class. Often times, the compilation task is delegated to a separate JSP compiler that will be responsible for identifying the necessary tag libraries, and generating the corresponding Servlet. The container then proceeds to load the dynamically generated class, creates an instance and dispatches the servlet request to that instance.

Supporting in-line compilation of a JSP inside a bundle will require that the Web Runtime maintains a private area where it can store such compiled classes. The Web Runtime can leverage its private bundle storage area. The Web Runtime can construct a special class loader to load generated JSP classes such that classes from the bundle class path are visible to newly compiled JSP classes.

The JSP specification does not describe how JSP pages are dynamically compiled or reloaded. Various Web Runtime implementations handle the aspects in proprietary ways. This specification does not bring forward any explicit requirements for supporting dynamic aspects of JSP pages.

**128.7 Security**

The security aspects of this specification are defined by the [3] Servlet 2.5 specification.

**128.8 References**


[2] Java EE Web Applications

[3] Servlet 2.5 specification
http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index.html

[4] JSP 2.1 specification
http://jcp.org/aboutJava/communityprocess/final/jsr245/index.html

[5] PAX Web Extender
http://team.ops4j.org/wiki/display/paxweb/Pax-Web

http://www.ietf.org/rfc/rfc1738.txt

[7] JSR 250 Common Annotations for the Java Platform
http://jcp.org/aboutJava/communityprocess/pfd/jsr250/index.html
130 Coordinator Service Specification

Version 1.0

130.1 Introduction

The OSGi programming model is based on the collaboration of standard and custom components. In such a model there is no central authority that has global knowledge of the complete application. Though this lack of authority can significantly increase reusability (and robustness) there are times when the activities of the collaborators must be coordinated. For example, a service that is repeatedly called in a task could optimize performance by caching intermediate results until it knew the task was ended.

To know when a task involving multiple collaborators has ended is the primary purpose of the Coordinator service specification. The Coordinator service provides a rendezvous for an initiator to create a Coordination where collaborators can decide to participate. When the Coordination has ended, all participants are informed.

This Coordinator service provides an explicit Coordination model, the Coordination is explicitly passed as a parameter, and an implicit model where the Coordination is associated with the current thread. Implicit Coordinations can be nested.

Coordinators share the coordination aspects of the resource model of transactions. However, the model is much lighter-weight because it does not support any of the ACID properties.

130.1.1 Essentials

- **Coordination** - Provide a solution to allow multiple collaborators to coordinate the outcome of a task initiated by an initiator.
- **Initiator** - An initiator must be able to initiate a coordination and control the final outcome.
- **Participants** - Participants in the task must be informed when the coordination has ended or failed as well as being able to terminate the Coordination.
- **Time-out** - A Coordination should fail after a given time-out.
- **Blocking** - Provide support for blocking and serializing access to Participants.
- **Nesting** - It must be possible to nest Coordinations.
- **Per Thread Model** - Provide a per-thread current Coordination model.
- **Variables** - Provide a variable space per Coordination

130.1.2 Entities

- **Coordinator** - A service that can create and enumerate Coordinations.
- **Coordination** - Represents the ongoing Coordination.
- **Initiator** - The party that initiates a Coordination.
- **Participant** - A party that wants to be informed of the outcome of a Coordination.
- **Collaborator** - Either a participant or initiator.
130.2 Usage

This section is an introduction in the usage of the Coordinator service. It is not the formal specification, the normative part starts at Coordinator Service on page 513. This section leaves out some of the details for clarity.

130.2.1 Synopsis

The Coordinator service provides a mechanism for multiple parties to collaborate on a common task without a priori knowledge of who will collaborate in that task. A collaborator can participate by adding a Participant to the Coordination. The Coordination will notify the Participants when the coordination is ended or when it is failed.

Each Coordination has an initiator that creates the Coordination object through the Coordinator service. The initiator can then push this object on a thread-local stack to make it an implicit Coordination or it can pass this object around as a parameter for explicit Coordinations. Collaborators can then use the current Coordination on the stack or get it from a parameter. Whenever a bundle wants to participate in the Coordination it adds itself to the Coordination as a participant. If necessary, a collaborator can initiate a new Coordination, which could be a nested Coordination for implicit Coordinations.

A Coordination must be terminated. Termination is either a normal end when the initiator calls the end method or it is failed when the fail method is called. A Coordination can be failed by any of the collaborators. A Coordination can also fail independently due to a time-out or when the initiator releases its Coordinator service. All participants in the Coordination are informed in reverse participation order about the outcome in a callback for ended or failed Coordinations.

A typical action diagram with a successful outcome is depicted in Figure 130.2.
130.2.2 Explicit Coordination

The general pattern for an initiator is to create a Coordination through the Coordinator service, perform the work in a try block, catch any exceptions and fail the Coordination in the catch block, and then ensure ending the Coordination in the finally block. The finally block can cause an exception. This is demonstrated in the following example:

```java
Coordination c = coordinator.create("com.example.work",0);
try {
    doWork(c);
} catch( Exception e ) {
    c.fail(e);
} finally {
    c.end();
}
```

This deceptively small template is quite robust:

- If the `doWork` method throws an Exception then the template fails with a Coordination Exception because it is failed in the try block.
- Any exceptions thrown in the try block are automatically causing the Coordination to fail.
- The Coordination is always terminated and removed from the stack due to the finally block.
- All failure paths, Coordinations that are failed by any of the collaborators, time-outs, or other problems are handled by the `end` method in the finally block. It will throw a `FAILED` or `PARTIALLY_ENDED` Coordination Exception for any of the failures.

The different failure paths and their handling is pictured in Figure 130.3.
The example shows an explicit Coordination because the create method is used. Implicit Coordinations are used in *Implicit Coordinations* on page 507. The parameters of the create method are the name of the Coordination and its time-out. The name is used for informational purposes as well as security. For security reasons, the name must follow the same syntax as the Bundle Symbolic Name. In a secure environment the name can be used to limit Coordinations to a limited set of bundles. For example, a set of bundles signed by a specific signer can use names like `com.acme.*` that are denied to all other bundles.

The zero time-out specifies that the Coordination will not have a time-out. Otherwise it must be a positive long, indicating the number of milliseconds the Coordination may take. However, implementations should have a configurable time-out to ensure that the system remains alive.

In the doWork method the real work is done in conjunction with the collaborators. Explicit Coordinations can be passed to other threads if needed. Collaborators can decide to add participants whenever they require a notification when the Coordination has been terminated. For example, the following code could be called from the doWork method:

```java
void foo(Coordination c) {
    doPrepare();
    c.addParticipant(this);
}
```

This method does the preparation work but does not finalize it so that next time it can use some intermediate results. For example, the prepare method could cache a connection to a database that should be reused during the Coordination. The collaborator can assume that it will be called back on either the failed or ended method. These methods could look like:

```java
public void ended(Coordination c) { doFinish(); }
public void failed(Coordination c) { doFailed(); }
```
The Coordinator provides the guarantee that this code will always call the `doFinish` method when the Coordination succeeds and `doFailed` method when it failed.

The Participant must be aware that the `ended(Coordination)` and `failed(Coordination)` methods can be called on any thread.

If the `doWork` method throws an exception it will end up in the catch block of the initiator. The catch block will then fail the Coordination by calling the `fail` method with the given exception. If the Coordination was already terminated because something else already had failed it then the method call is ignored, only the first fail is used, later fails are ignored.

In all cases, the finally block is executed last. The finally block ends the Coordination. If this coordination was failed then it will throw a Coordination Exception detailing the reason of the failure. Otherwise it will terminate it and notify all the participants.

The Coordination Exception is a Runtime Exception making it unnecessary to declare it.

### 130.2.3 Multi Threading

Explicit Coordinations allow the Coordination objects to be passed to many different collaborators who can perform the work on different threads. Each collaborator can fail the Coordination at any moment in time or the time-out can occur on yet another thread. Participants must therefore be aware that the callbacks `ended` and `failed` can happen on any thread. The following example shows a typical case where a task is parallelized. If any thread fails the Coordination, all other threads could be notified before they're finished.

```java
Executor executor = ...;
final CountDownLatch latch = new CountDownLatch(10);
final Coordination c = coordinator.create("parallel", 0);
for (int i=0; i<10; i++) {
    executor.execute(
        new Runnable() {
            public void run() { baz(c); latch.countDown(); }
        }));
    latch.await();
    c.end();
}
```

The Coordination object is thread safe so it can be freely passed around.

### 130.2.4 Implicit Coordinations

An explicit Coordination requires that the Coordination is passed as a parameter to the `doWork` method. The Coordinator also supports implicit Coordinations. With implicit Coordinations the Coordinator maintains a thread local stack of Coordinations where the top of this stack is the current Coordination for that thread. The usage of the implicit Coordination is almost identical to the explicit Coordinations except that all the work occurs on a single thread. The control flow is almost identical to explicit Coordinations:

```java
Coordinator c = coordinator.begin("com.example.work", 0);
try {
    doWork();
} catch (Exception e) {
    c.fail(e);
} finally {
    c.end();
}
```
See also Figure 130.3. However, in this case the finally block with the call to the end method is even more important. With an implicit Coordination the Coordination is put on a thread local stack in the begin method and must therefore be popped when the Coordination is finished. The finally block ensures therefore the proper cleanup of this thread local stack.

The difference between implicit and explicit Coordinations is that the implicit Coordination is not passed as a parameter, instead, collaborators use the current Coordination. With implicit Coordinations all method invocations in a thread can always access the current Coordination, even if they have many intermediates on the stack. The implicit model allows a collaborator many levels down the stack to detect a current Coordination and register itself without the need to modify all intermediate methods to contain a Coordination parameter. The explicit model has the advantage of explicitness but requires all APIs to be modified to hold the parameter. This model does not support passing the parameter through layers that are not aware of the Coordination. For example, OSGi services in general do not have a Coordination parameter in their methods making the use of explicit Coordinations impossible.

Collaborators can act differently in the presence of a current Coordination. For example, a collaborator can optimize its work flow depending on the presence of a current Coordination.

Coordinator coordinator = ...;
void foo() {
    doPrepare();
    if (!coordinator.addParticipant(this))
        doFinish();
}

The Coordinator service has an addParticipant method that makes working with the current Coordination simple. If there is a current Coordination then the Coordinator service will add the participant and return true, otherwise it returns false. It is therefore easy to react differently in the presence of a current Coordination. In the previous example, the doFinish method will be called immediately if there was no current Coordination, otherwise it is delayed until the Coordination fails or succeeds. The participant callbacks look the same as in the previous section:

    public void ended(Coordination c) { doFinish(); }
    public void failed(Coordination c) { doFailed(); }

Though the code looks very similar for the implicit and explicit Coordinations there are some additional rules for implicit Coordinations.

The end method must be called on the same thread as the begin method, trying to end it on another thread results in a WRONG_THREAD Coordination Exception being thrown.

Even though the end method must be called on the initiating thread, the callbacks to the Participants can be done on any thread as the specification allows the Coordinator to use multiple threads for all callbacks.

### 130.2.5 Partial Ending

The Coordination is a best effort mechanism to coordinate, not a transaction model with integrity guarantees. This means that users of the Coordinator service must understand that there are cases where a Coordination ends in limbo. This happens when one of the Participants throws an Exception in the ended callback. This is similar to a transactional resource manager failing to commit in a 2-phase commit after it has voted yes in the prepare phase; a problem that is the cause of much of the complexity of a transaction manager. The Coordinator is limited to use cases that do not require full ACID properties and can therefore be much simpler. However, users of the Coordinator service must be aware of this limitation.

If a Participant throws an exception in the ended method, the end method that terminated the Coordination must throw a PARTIALLY_ENDED Coordination Exception. It is then up to the initiator to
correct the situations. In most cases, this means allowing the exception to be re-thrown and handle the failure at the top level. Handling in those cases usually implies logging and continuing.

The following code shows how the `PARTIALLY_ENDED` case can be handled more explicitly.

```java
Coordinator c = coordinator.begin("work",0);
try{
doWork();
} catch( Exception e ) {
c.fail(e);
} finally {
try{
c.end();
} catch( CoordinationException e ) {
if ( e.getType() == CoordinationException.PARTIALLY_ENDED) {
// limbo!
...
}
}
}
```

### 130.2.6 Locking

To participate in a Coordination and receive callbacks a collaborator must add a `Participant` object to the Coordination. The `addParticipant(Participant)` method blocks if the given `Participant` object is already used in another Coordination. This blocking facility can be used to implement a number of simple locking schemes that can simplify maintaining state in a concurrent environment.

Using the `Participant` object as the key for the lock makes it simple to do course grained locking. For example, a service implementation could use the service object as a lock, effectively serializing access to this service when it is used in a Coordination. Coarse grained locking allows all the state to be maintained in the coarse object and not having to worry about multiplexing simultaneous requests. The following code uses the coarse locking pattern because the collaborator implements the `Participant` interface itself:

```java
public class Collaborator implements Participant{
    public void doWork(Coordination coordination ) {
        ...
        coordination.addParticipant(this);
    }

    public void ended(Coordination c) { ... }
    public void failed(Coordination c) { ... }
}
```

The simplicity of the coarse grained locking is at the expense of lower performance because tasks are serialized even if it would have no contention. Locks can therefore also be made more fine grained, allowing more concurrency. In the extreme case, creating a new object for each participation makes it possible to never lock. For example, the following code never locks because it always creates a new object for the `Participant`:

```java
public void doWork(Coordination coordination){
    final State state = ...  
    coordination.addParticipant(new Participant() {
        public void ended(Coordination c) { state ... } 
        public void failed(Coordination c) { state ... }
    });
```
130.2.7 Failing

Any collaborator can fail an ongoing Coordination by calling the `fail(Throwable)` method, the Throwable parameter must not be null. When the Coordination has already terminated then this is a no-op. The Coordinator service has a convenience method that fails the current Coordination if present. The fail methods return a boolean that is true when the method call causes the termination of the Coordination, in all other cases it is false.

Failing a Coordination will immediately perform the callbacks and reject any additional Participants by throwing an `ALREADY_ENDED` Coordination Exception. The asynchronous nature of the fail method implies that it is possible to have been called even before the `addParticipant(Participant)` method has returned. Anybody that has the Coordination object can check the failed state with the `getFailure()` method.

In general, the best and most robust strategy to handle failures is to throw an Exception from the collaborator, allowing the initiator to catch the exception and properly fail the Coordination.

130.2.8 Time-out

The time-out is specified in the Coordinator `create(String, long)` or `begin(String, long)` methods. A time-out of zero is indefinite, otherwise the time-out specifies the number of milliseconds the Coordination can take to terminate. A given time-out can be extended with the `extendTimeout(long)` method. This method will add an additional time-out to the existing deadline if a prior deadline was set. For example, the following code extends the time-out with 5 seconds whenever a message must be sent to a remote address:

```java
Object sendMessage(Message m) {
    Coordination c = coordinator.peek();
    Address a = m.getDestination();
    if (c != null && a.isRemote()) {
        c.extendTimeout(5000);
    }
    return sendMessage0(m);
}
```

Applications should not rely on the exact time-out of the Coordination and only use it as a safety function against deadlocks and hanging collaborators.

130.2.9 Joining

When a Coordination is terminated it is not yet completely finished, the callback to the Participants happens after the atomic termination. In certain cases it is necessary to ensure that a method does not progress until all the participants have been notified. It is therefore possible to wait for the Coordination to completely finish with the `join(long)` method. This method can have a time-out. For example:

```java
void collaborate( final Coordination c ) {
    doWork();
    Thread t = new Thread() {
        public void run(){
            try {
                c.join(0);
                ... // really terminated here, all participants called back
            } catch( Exception e) { ... }
        }
    };
```
130.2.10 Variables

A Participant is likely to have to maintain state that is particular for the collaboration. This state is usually needed in the ended method to properly finalize the work. In general, the best place to store this state is in the Participant object itself, inner classes and final variables are a good technique for storing the state. However, the state can also be stored in a Coordination variable. Each Coordination has a private set of variables that can be obtained with the `getVariables()` method. The resulting map takes a class as the key and returns an Object. The map is not synchronized, any changes to the map must be synchronized on the returned Map object to ensure the visibility of the changes to other threads. The class used for the key is not related to the returned type, it is a Class object to provide a convenient namespace.

The following example shows how the state can be stored with variables.

```java
public void doWork(Coordination coordination) {
    Map<Class<?>, Object> map = coordination.getVariables();
    synchronized(map) {
        State state = (State) map.get(SharedWorker.class);
        if (state == null) {
            state = new State(this);
            map.put(state);
            ... do initial work
        }
        ... do other work
        coordination.addParticipant(this);
    }
}

public void ended(Coordination c) {
    Map<Class<?>, Object> map = coordination.getVariables();
    synchronized(map) {
        State state = (State) map.get(SharedWorker.class);
        .. finalize
    }
}

public void failed(Coordination c) {
    Map<Class<?>, Object> map = coordination.getVariables();
    synchronized(map) {
        State state = (State) map.get(SharedWorker.class);
        .. finalize
    }
}
```

130.2.11 Optimizing Example

For example, a web based system has a charge service:

```java
public interface Charge {
    void charge(String reason, int amount);
}
```

This service is used throughout the system for charging the tasks the system performs. Each servlet request can actually create multiple Charge Data Records (CDR). For this reason, a Coordination is started before the page is constructed. Each part of the page that has an associated cost must create a CDR. There are the following issues at stake:
Charging should not take place when failing, and
Performance can be optimized to only persist the CDRs once, and
The user must be passed to the Charge service.

To begin with the request code:

```java
public void doGet(HttpServletRequest rq, HttpServletResponse rsp) {
    Coordination c = coordinator.begin("com.acme.request", 30000);
    try {
        Principal p = rq.getUserPrincipal();
        Map<Class<?>, Object> map = c.getVariables();
        map.put(Principal.class, p);
        buildPage(rq, rsp);
    } catch(Exception e) {
        c.fail(e);
    }
    finally {
        c.end();
    }
}
```

Each method that has a charge will call the Charge service. The following code shows an implementation of this Charge service.

```java
public class ChargeImpl implements Charge, Participant {
    final List<CDR> records = new ArrayList<CDR>();

    public void charge(String reason, int amount) {
        Coordination c = coordinator.peek();
        if (c == null) {
            save(Arrays.asList(new CDR(null, reason, amount)));
        } else {
            Principal p = getPrincipal(c);
            records.add(new CDR(p, reason, amount));
            c.addParticipant(this);
        }
    }

    Principal getPrincipal(Coordination c) {
        if (c == null)
            return null;
        Map<Class<?>, Object> map = c.getVariables();
        synchronized(map) {
            Principal p = (Principal) map.get(Principal.class);
            return p != null ? p : getPrincipal(c.getEnclosingCoordination());
        }
    }

    public void ended(Coordination c) {
        save(records);
        records.clear();
    }

    public void failed(Coordination c) {
        records.clear();
    }
}
```

void save(List<CDR> records) { ... }
130.2.12 Security Example

The Coordination Permission is a filter based permission that is asserted for many of the methods in the API, the bundle that is checked is always the bundle that created the corresponding Coordination. For example:

```
ALLOW {
    [ BundleSignerCondition "cn=ACME" ]
    { CoordinationPermission "(signer=cn=ACME)" "*" }
}
```

This example allows bundles signed by ACME to perform all Coordination actions on Coordinations created by bundles signed by ACME.

The filter can also assert the name of the Coordination:

```
coordination.name
```

It is therefore possible to create a name based protection scheme. By denying all bundles except a select group through the use of a name prefix, the use of Coordinations can be restricted to this select group:

```
DENY {
    [ BundleSignerCondition "cn=ACME" "!" ]
    { CoordinationPermission "(coordination.name=com.acme.*)" "*" }
}
 ALLOW {
    { CoordinationPermission "(coordination.name=*)" "*" }
}
```

If a bundle is not signed by ACME it will be denied the use of Coordination names starting with com.acme. though it will be allowed to use any other name. This effectively enables only bundles signed by ACME to create Coordinations with this name prefix.

130.3 Coordinator Service

The Coordinator service is the entry point for the Coordination. It provides the following functions:

- Coordination creation
- Life cycle management of a Coordination
- Thread based Coordinations
- Introspection

130.3.1 Coordination Creation

A Coordination object is created by an *initiator*. An initiator can create a Coordination object with the Coordinator `create(String,long)` or `begin(String,long)` method. Each Coordination when created gets a positive long identity that is available with `getId()`. Ids are a unique identifier for a specific Coordinator service. The id is always increasing, that is, a Coordination with a higher id is created later.

The create methods specify the name of the Coordination. This name is a security concept, see Security on page 518, as well as used for debugging. The coordination name must therefore conform to the same syntax as a bundle symbolic name:

```
coordination-name ::= symbolic-name   // see OSGi Core Release 6
```
Passing a name that does not conform to this syntax must throw an Illegal Argument Exception. There are no constraints on duplicates, multiple different Coordinations can use the same name. The name of the Coordination is available with the `getName()` method.

### 130.3.2 Adding Participants

The Coordination object can be passed to collaborators as a parameter in a method call. Some of these collaborators might be interested in participating in the given Coordination, they can achieve this by adding a Participant object to the Coordination.

A Participant is a collaborator that requires a callback after the Coordination has been terminated, either when it ended or when it failed. To participate, it must add a Participant object to a Coordination with the `addParticipant(Participant)` method on Coordination. This method throws an `ALREADY_ENDED` or `FAILED` Coordination Exception when the Coordination has been terminated.

When a Participant is:

- **Not in any Coordination** - Add it to the given Coordination and return.
- **In target Coordination** - Ignore, participant is already present. A Participant can participate in the same Coordination multiple times by calling `addParticipant(Participant)` but will only be called back once when the Coordination is terminated. Its order must be defined by the first addition.
- **In another Coordination** - Lock until after the other Coordination has notified all the Participants. Implementations can detect deadlocks in certain cases and throw a Coordination Exception if a dead lock exist, otherwise the deadlock is solved when the Coordination times out.

Verifying if a Participant object is already in another Coordination must use identity and not equality.

### 130.3.3 Active

A Coordination is active until it is terminated. A Coordination can terminate because it is ended, or it is failed. The following methods cause a termination:

- **end()** - A normal end. All participants that were added before the end call are called back on their `ended(Coordination)` method.
- **fail(Throwable)** - The Coordination has failed, this will call back the `failed(Coordination)` method on the participants. This method can be called by the Coordinator, the initiator, or any of the collaborators. There are a number of failures that are built in to the Coordinator. These failures use singleton Exception instances defined in the Coordination interface:
  - **TIMEOUT** - If the Coordination times out the Coordination is failed with the `TIMEOUT` exception instance in Coordination.
  - **RELEASED** - If the Coordinator that created the Coordination was unget, all Coordinations created by it will fail with the `RELEASED` exception.

The state diagram for the Coordination is pictured in Figure 130.4.

*Figure 130.4 Coordination state diagram*
130.3.4 Explicit and Implicit Models

The Coordinator supports two very different models of usage: explicit and implicit. The explicit model is when a Coordination is created and passed around as a parameter. The second model is the implicit model where the Coordinator maintains a thread local stack of Coordinations. Any collaborator can then decide to use the top of the stack as the current Coordination. The peek() method provides access to the current Coordination.

The begin(String,long) method creates a new Coordination and pushes this on the stack, beginning an implicit Coordination. This is identical to:

```java
coordinator.create("work",0).push();
```

Once a Coordination is pushed on a stack it is from that moment on associated with the current thread. A Coordination can only be pushed once, the ALREADY_PUSHED Coordination Exception must be thrown when the Coordination is already associated with one of the thread local stacks maintained by the Coordinator service.

The Coordination is removed from the stack in the end() method. The end() method must not only terminate itself but it must also terminate all nested Coordinations. The current Coordination can also be explicitly removed with the Coordinator pop() method.

A Coordination that is pushed on a thread local stack returns the associated thread on the getThread() method. This method returns null for Coordinations not on any stack, that is, explicit Coordinations.

130.3.5 Termination

Both the end() and fail(Throwable) methods terminate the Coordination if it was not already terminated. Termination is atomic, only the end or the fail method can terminate the Coordination. Though this happens on different threads, a Coordination can never both end and fail from any perspective. That is, if a fail races with end then only one of them can win and the other provides the feedback that the Coordination was already terminated.

Terminating a Coordination has the following effects:

- It is atomic, it can only happen once in a Coordination
- It freezes the set of participants, no more participants can be added

130.3.6 Ending

The end() method should always be called at the end of a Coordination to ensure proper termination, notification, and cleanup. The end method throws a FAILED or PARTIALLY_ENDED Coordination Exception if the Coordination was failed before.

If the Coordination had already been ended before then this is a programming error and an ALREADY_ENDED Configuration Exception is thrown. The end() method should never be called twice on the same Coordination.

If the termination succeeds then the participants must be notified by calling the ended(Coordination) method on each Participant that had been successfully added to the Coordination. This callback can take place on any thread but must be in reverse order of adding. That is, the last added Participant is called back first.

Participants must never make any assumptions about the current Coordination in the callback. The Coordination it was added to is therefore given as an explicit parameter in the ended(Coordination) method.

If a Participant throws an Exception then this must not prevent the calling of the remaining participants. The Exception should be logged. If a Participant has thrown an Exception then the end()
method must throw a PARTIALLY_ENDED Coordination Exception after the last Participant has returned from its callback, otherwise the method returns normally. Participants should normally not throw Exceptions in their callbacks.

If the Coordination is implicit (it is pushed on a stack) then the Coordination must be removed from its stack after the participants have been called back. This requires that the ending thread is the same as the thread of the Coordination. The end thread is the thread of the end() method call. If the Coordination’s thread is not the same as the ending thread then a WRONG_THREAD Coordination Exception is thrown.

If the ending Coordination is on the stack but it is not the current Coordination then each nested Coordination must be ended before the current Coordination, see Nesting Implicit Coordinations on page 516 for more information.

The fail(Throwable) method must not remove the current Coordination, it must remain on the stack. The initiator must always call the end() method. Always calling end() in a finally block is therefore paramount.

130.3.7 Failing, TIMEOUT, ORPHANED, and RELEASED

Failing can happen asynchronously during the time a Coordination is active. A Coordination is failed by calling fail(Throwable). The Throwable argument must not be null, it is the cause of the failure.

Failing a Coordination must first terminate it. If the Coordination was already terminated the fail(Throwable) method has no effect. Otherwise, it must callback all its added Participants on the failed(Coordination) callback method. Exceptions thrown from this method should be logged and further ignored. The callback can occur on any thread, including the caller’s.

Implicit Coordinations must not be popped from its stack in a fail nor is it necessary to call the fail method from any particular thread. The removal of the Coordination from the stack must happen in the end method.

There are two asynchronous events that can also fail the Coordination. If the Coordination times out, it will be treated as a fail( TIMEOUT ) and if the Coordinator is ungotten with active Coordinations then each of those Coordinations must fail as if fail( RELEASED ) is called.

A Coordination can also be orphaned. An orphaned Coordination has no longer any outside references. This means that the Coordination can no longer be ended or failed. Such Coordinations must fail with an ORPHANED Exception.

130.3.8 Nesting Implicit Coordinations

Implicit Coordinations can be nested. For this reason, the Coordinator maintains a thread local stack of Coordinations where the top, accessible with the peek() method, is the current Coordination. Each time a new Coordination is begun with the begin(String,long) method, the current Coordination is replaced with the newly created Coordination. When that Coordination is ended, the previous current Coordination is restored. Nesting is always on the same thread, implicit Coordinations are always associated with a single thread, available through its getThread() method. The end method must be called on the same thread as the begin(String,long) or last push() method.

Using the standard model for implicit Coordinations, where the initiator always ends the Coordination on the same thread as it begun, ensures that nesting is properly handled. However, in certain cases it is necessary to manipulate the stack or make implicit Coordinations explicit or vice versa. For this reason, it is possible to pop Coordinations from the stack with the pop() method. This method disassociates the Coordination from the current thread and restores the previous (if any) Coordination as the current Thread. A Coordination can then be made the current Coordination for a thread by calling the push() method. However, a Coordination can be pushed on the stack at most once. If a Coordination is pushed a second time, in any thread, the ALREADY_PUSHED Coordination Exception must be thrown.
The Coordination is removed from its stack when the `end()` method is called. It is therefore highly recommended to always end a Coordination in the nesting order. However, it is possible that a Coordination is ended that is not the current Coordination, it has nested Coordinations that were not properly ended. In that case all nested Coordinations must be ended in reverse creation order, that is, the current Coordination first, by calling the `end` method on it.

If any Coordination fails to end properly (including `PARTIALLY_ENDED`) then the remaining Coordinations on the stack must fail and chain the exceptions. In pseudo code:

```java
while (coordinator.peek() != this) {
    try {
        coordinator.peek().end();
    } catch (CoordinationException e) {
        coordinator.peek().fail(e);
    }
}
```

### 130.3.9 Time-outs

When a Coordination is created it will receive a time-out. A time-out is a positive value or zero. A zero value indicates that the Coordination should have no time-out. This does not imply that a Coordination will never time-out, implementations are allowed to be configured with a limit to the maximum active time for a Coordination.

Collaborators can extend the time out with the `extendTimeout(long)` method. If no time-out was set (0), this method will be ignored. Otherwise the given amount (which must be positive) is added to the existing deadline. A Coordinator implementation can fail the Coordination earlier, however, when configured to do so.

If a Coordination is timed out, the Coordination is failed with a `fail(TIMEOUT)` method call from an unspecified thread, see *Failing, TIMEOUT, ORPHANED, and RELEASED* on page 516.

### 130.3.10 Released

The Coordination's life cycle is bound to the Coordinator service that created it. If the initiator's bundle ungets this service then the Coordinator must fail all the Coordinations created by this Coordinator by calling the `fail(RELEASED)` method.

Participants from bundles that are stopped are not taken into account. This means that it is possible that a participant is called while its bundle is stopped. Stopped Participants should fail any Coordinations that they participate in.

### 130.3.11 Coordinator Convenience Methods

The Coordinator contains a number of convenience methods that can be used by collaborators to interact with the current Coordination.

- `begin(String,long)` - Is logically the same as `create(String,long).push()`.
- `addParticipant(Participant)` - This method makes it easy to react differently to the presence of a current implicit Coordination. If a current Coordination exists, the participant is added and true is returned (or an exception thrown if the Coordination is already terminated), otherwise false is returned.
- `fail(Throwable)` - If there is no current Coordination, this method returns false. Otherwise it returns the result of calling `fail(Throwable)` on the current Coordination. This method therefore only returns true when a current Coordination was actually terminated due to this call.

### 130.3.12 Administrative Access

The Coordination objects provide a number of methods that are used for administrating the Coordinations and the Coordinator.
• **getBundle()** - Provide the bundle that created the Coordination. This bundle is the bundle belonging to the Bundle Context used to get the Coordinator service.

• **getFailure()** - The Exception that caused this Coordination to fail or null. There are two fixed exception instances for a time out (`TIMEOUT`), when the Coordination is orphaned (`ORPHANED`), and when the Coordinator service is released (`RELEASED`).

• **getId()** - The Coordination's id.

• **getName()** - The name of the Coordination.

• **getParticipants()** - The current list of participants. This is a mutable snapshot of the added participants. Changing the snapshot has no effect on the Coordination.

• **getThread()** - Answer the thread associated with an implicit Coordination. If the Coordination is not implicit then the answer is null.

• **getEnclosingCoordination()** - Return the enclosing Coordination.

And for the Coordinator:

• **getCoordination(long)** - Retrieve a Coordination by its id.

• **getCoordinations()** - Get a list of active Coordinations

### 130.3.3 Summary

A Coordination can exist in three different states **ACTIVE**, **END**, and **FAIL**. During its life it will transition from **ACTIVE** to either **END** or **FAIL**. The entry (when the state is entered) and exit (when the state is left) actions when this transition takes place and the effect on the different methods are summarized in the following table.

<table>
<thead>
<tr>
<th>State/Method</th>
<th><strong>ACTIVE</strong></th>
<th><strong>END</strong></th>
<th><strong>FAIL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>entry action</td>
<td>Notify all the participants by calling the <code>ended(Coordination)</code> method.</td>
<td>Notify all the participants by calling the <code>failed(Coordination)</code> method.</td>
<td></td>
</tr>
<tr>
<td>exit action</td>
<td>Terminate</td>
<td>throws <strong>ALREADY_ENDED</strong></td>
<td>throws <strong>FAILED</strong></td>
</tr>
<tr>
<td>end()</td>
<td>➔ <strong>END</strong>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fail(Throwable)</td>
<td>➔ <strong>FAIL</strong>, return true.</td>
<td>return false.</td>
<td>return false.</td>
</tr>
<tr>
<td>PARTIALLY_ENDED</td>
<td>Can throw</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 130.4 Security

This specification provides a Coordination Permission. This permission can enforce the name of the coordination as well as assert the properties of the initiating bundle, like for example the signer or bundle symbolic name. The permission therefore uses a filter as name, as defined in the filter based permissions section in *OSGi Core Release 6*, see *OSGi Core Release 6*. There is one additional parameter for the filter:

```
coordination.name
```

The value is the given name of the Coordination. Restricting the name of a Coordination allows the deployer to limit the use of this name to a restricted set of bundles.

The following actions are defined:

• **INITIATE** - Required to initiate and control a Coordination.

• **PARTICIPATE** - Required to participate in a Coordination.
- **ADMIN** - Required to administrate a Coordinator.

The target bundle of the Coordination Permission is the initiator's bundle. This is the bundle that got the Coordinator service to create the Coordination. An initiator must therefore have permission to create Coordinations for itself.

There are two constructors available:

- **CoordinationPermission(String,String)** - The constructor for the granted permission. It is given a filter expression and the actions that the permission applies to.
- **CoordinationPermission(String,Bundle,String)** - The constructor for the requested permission. It is given the name of the permission, the bundle that created the corresponding coordination, and the requested actions.

### org.osgi.service.coordinator

Coordinator 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.coordinator; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.coordinator; version="[1.0,1.1)"
```

### Summary

- **Coordination** - A Coordination object is used to coordinate a number of independent Participants.
- **CoordinationException** - Unchecked exception which may be thrown by a Coordinator implementation.
- **CoordinationPermission** - A bundle's authority to create or use a Coordination.
- **Coordinator** - A Coordinator service coordinates activities between different parties.
- **Participant** - A Participant participates in a Coordination.

### public interface Coordination

A Coordination object is used to coordinate a number of independent Participants.

Once a Coordination is created, it can be used to add Participant objects. When the Coordination is ended, the participants are notified. A Coordination can also fail for various reasons. When this occurs, the participants are notified of the failure.

A Coordination must be in one of two states, either ACTIVE or TERMINATED. The transition between ACTIVE and TERMINATED must be atomic, ensuring that a Participant can be guaranteed of either receiving an exception when adding itself to a Coordination or of receiving notification the Coordination has terminated.

A Coordination object is thread safe and can be passed as a parameter to other parties regardless of the threads these parties use.

The following example code shows how a Coordination should be used.

```java
void foo() {
```
Coordination c = coordinator.create("work", 0);
try {
    doWork(c);
} catch (Exception e) {
    c.fail(e);
} finally {
    c.end();
}

Concurrency
Thread-safe

Provider Type
Consumers of this API must not implement this type

130.5.2.1 public static final Exception ORPHANED
A singleton exception that will be the failure cause when a Coordination has been orphaned.

130.5.2.2 public static final Exception RELEASED
A singleton exception that will be the failure cause when the Coordinations created by a bundle are terminated because the bundle released the Coordinator service.

130.5.2.3 public static final Exception TIMEOUT
A singleton exception that will be the failure cause when a Coordination times out.

130.5.2.4 public void addParticipant(Participant participant)
participant
The Participant to register with this Coordination. The participant must not be null.

□ Register a Participant with this Coordination.

Once a Participant is registered with this Coordination, it is guaranteed to receive a notification for either normal or failure termination when this Coordination is terminated.

Participants are registered using their object identity. Once a Participant is registered with this Coordination, subsequent attempts to register the Participant again with this Coordination are ignored and the Participant is only notified once when this Coordination is terminated.

A Participant can only be registered with a single active Coordination at a time. If a Participant is already registered with an active Coordination, attempts to register the Participation with another active Coordination will block until the Coordination the Participant is registered with terminates.

Notice that in edge cases the notification to the Participant that this Coordination has terminated can happen before this method returns.

Attempting to register a Participant with a terminated Coordination will result in a CoordinationException being thrown.

The ordering of notifying Participants must follow the reverse order in which the Participants were registered.

Throws CoordinatonException – If the Participant could not be registered with this Coordination. This exception should normally not be caught by the caller but allowed to be caught by the initiator of this Coordination.

SecurityException – If the caller does not have CoordinationPermission[PARTICIPATE] for this Coordination.

130.5.2.5 public void end()
□ Terminate this Coordination normally.
If this Coordination has been pushed on the thread local Coordination stack of another thread, this method does nothing except throw a CoordinationException of type CoordinationException.WRONG_THREAD.

If this Coordination has been pushed on the thread local Coordination stack of this thread but is not the current Coordination, then the Coordinations on the thread local Coordination stack above this Coordination must be terminated and removed from the thread local Coordination stack before this Coordination is terminated. Each of these Coordinations, starting with the current Coordination, will be terminated normally. If the termination throws a CoordinationException, then the next Coordination on the thread local Coordination stack will be terminated as a failure with a failure cause of the thrown CoordinationException. At the end of this process, this Coordination will be the current Coordination and will have been terminated as a failure if any of the terminated Coordinations threw a CoordinationException.

If this Coordination is the current Coordination, then it will be removed from the thread local Coordination stack.

If this Coordination is already terminated, a CoordinationException is thrown. If this Coordination was terminated as a failure, the failure cause will be the cause of the thrown CoordinationException.

Otherwise, this Coordination is terminated normally and then all registered Participants are notified. Participants should finalize any work associated with this Coordination. The successful return of this method indicates that the Coordination has terminated normally and all registered Participants have been notified of the normal termination.

It is possible that one of the Participants throws an exception during notification. If this happens, this Coordination is considered to have partially failed and this method must throw a CoordinationException of type CoordinationException.PARTIALLY_ENDED after all the registered Participants have been notified.

Throws

CoordinationException—If this Coordination has failed, including timed out, or partially failed or this Coordination is on the thread local Coordination stack of another thread.

SecurityException—If the caller does not have CoordinationPermission[INITIATE] for this Coordination.

130.5.2.6  public long extendTimeout(long timeMillis)

timeMillis  The time in milliseconds to extend the current timeout. If the initial timeout was specified as 0, no extension must take place. A zero must have no effect.

☐ Extend the time out of this Coordination.

Participants can call this method to extend the timeout of this Coordination with at least the specified time. This can be done by Participants when they know a task will take more than normal time.

This method will return the new deadline if an extension took place or the current deadline if, for whatever reason, no extension takes place. Note that if a maximum timeout is in effect, the deadline may not be extended by as much as was requested, if at all. If there is no deadline, zero is returned. Specifying a timeout extension of 0 will return the existing deadline.

Returns

The new deadline in milliseconds. If the specified time is 0, the existing deadline is returned. If this Coordination was created with an initial timeout of 0, no timeout is set and 0 is returned.

Throws

CoordinationException—If this Coordination is terminated.

IllegalArgumentException—If the specified time is negative.

SecurityException—If the caller does not have CoordinationPermission[PARTICIPATE] for this Coordination.
130.5.2.7 public boolean fail(Throwable cause)

cause The failure cause. The failure cause must not be null.

☐ Terminate this Coordination as a failure with the specified failure cause.

If this Coordination is already terminated, this method does nothing and returns false.

Otherwise, this Coordination is terminated as a failure with the specified failure cause and then all registered Participants are notified. Participants should discard any work associated with this Coordination. This method will return true.

If this Coordination has been pushed onto a thread local Coordination stack, this Coordination is not removed from the stack. The creator of this Coordination must still call end() on this Coordination to cause it to be removed from the thread local Coordination stack.

Returns true if this Coordination was active and was terminated by this method, otherwise false.

Throws SecurityException – If the caller does not have CoordinationPermission[PARTICIPATE] for this Coordination.

130.5.2.8 public Bundle getBundle()

☐ Returns the bundle that created this Coordination. This is the bundle that obtained the Coordinator service that was used to create this Coordination.

Returns The bundle that created this Coordination.

Throws SecurityException – If the caller does not have CoordinationPermission[ADMIN] for this Coordination.

130.5.2.9 public Coordination getEnclosingCoordination()

☐ Returns the Coordination enclosing this Coordination if this Coordination is on the thread local Coordination stack.

When a Coordination is pushed onto the thread local Coordination stack, the former current Coordination, if any, is the enclosing Coordination of this Coordination. When this Coordination is removed from the thread local Coordination stack, this Coordination no longer has an enclosing Coordination.

Returns The Coordination enclosing this Coordination if this Coordination is on the thread local Coordination stack or null if this Coordination is not on the thread local Coordination stack or has no enclosing Coordination.

Throws SecurityException – If the caller does not have CoordinationPermission[ADMIN] for this Coordination.

130.5.2.10 public Throwable getFailure()

☐ Returns the failure cause of this Coordination.

If this Coordination has failed, then this method will return the failure cause.

If this Coordination timed out, this method will return TIMEOUT as the failure cause. If this Coordination was active when the bundle that created it released the Coordinator service, this method will return RELEASED as the failure cause. If the Coordination was orphaned, this method will return ORPHANED as the failure cause.

Returns The failure cause of this Coordination or null if this Coordination has not terminated as a failure.

Throws SecurityException – If the caller does not have CoordinationPermission[INITIATE] for this Coordination.
130.5.2.11 public long getId()

- Returns the id assigned to this Coordination. The id is assigned by the Coordinator service which created this Coordination and is unique among all the Coordinations created by the Coordinator service and must not be reused as long as the Coordinator service remains registered. The id must be positive and monotonically increases for each Coordination created by the Coordinator service.

  Returns The id assigned to this Coordination.

130.5.2.12 public String getName()

- Returns the name of this Coordination. The name is specified when this Coordination was created.

  Returns The name of this Coordination.

130.5.2.13 public List<Participant> getParticipants()

- Returns a snapshot of the Participants registered with this Coordination.

  Returns A snapshot of the Participants registered with this Coordination. If no Participants are registered with this Coordination, the returned list will be empty. The list is ordered in the order the Participants were registered. The returned list is the property of the caller and can be modified by the caller.

  Throws SecurityException – If the caller does not have CoordinationPermission[INITIATE] for this Coordination.

130.5.2.14 public Thread getThread()

- Returns the thread in whose thread local Coordination stack this Coordination has been pushed.

  Returns The thread in whose thread local Coordination stack this Coordination has been pushed or null if this Coordination is not in any thread local Coordination stack.

  Throws SecurityException – If the caller does not have CoordinationPermission[ADMIN] for this Coordination.

130.5.2.15 public Map<Class<?>,Object> getVariables()

- Returns the variable map associated with this Coordination. Each Coordination has a map that can be used for communicating between different Participants. The key of the map is a class, allowing for private data to be stored in the map by using implementation classes or shared data by using shared interfaces. The returned map is not synchronized. Users of the map must synchronize on the Map object while making changes.

  Returns The variable map associated with this Coordination.

  Throws SecurityException – If the caller does not have CoordinationPermission[PARTICIPANT] for this Coordination.

130.5.2.16 public boolean isTerminated()

- Returns whether this Coordination is terminated.

  Returns true if this Coordination is terminated, otherwise false if this Coordination is active.

130.5.2.17 public void join(long timeMillis) throws InterruptedException

timeMillis Maximum time in milliseconds to wait. Specifying a time of 0 will wait until this Coordination is terminated.

- Wait until this Coordination is terminated and all registered Participants have been notified.

  Throws InterruptedException – If the wait is interrupted.

  IllegalArgumentException – If the specified time is negative.
SecurityException – If the caller does not have CoordinationPermission[PARTICIPATE] for this Coordination.

public Coordination push()

Push this Coordination object onto the thread local Coordination stack to make it the current Coordination.

Returns This Coordination.

Throws CoordinationException – If this Coordination is already on the any thread’s thread local Coordination stack or this Coordination is terminated.

SecurityException – If the caller does not have CoordinationPermission[INITIATE] for this Coordination.

public class CoordinationException extends RuntimeException

Unchecked exception which may be thrown by a Coordinator implementation.

public static final int ALREADY_ENDED = 4

The Coordination has already terminated normally.

public static final int ALREADY_PUSHED = 5

The Coordination was already on a thread’s thread local Coordination stack.

public static final int DEADLOCK_DETECTED = 1

Registering a Participant with a Coordination would have resulted in a deadlock.

public static final int FAILED = 2

The Coordination has terminated as a failure with Coordination.fail(Throwable). When this exception type is used, the getCause() method must return a non-null value.

public static final int LOCK_INTERRUPTED = 6

The current thread was interrupted while waiting to register a Participant with a Coordination.

public static final int PARTIALLY_ENDED = 3

The Coordination has partially ended.

public static final int UNKNOWN = 0

Unknown reason for this exception.

public static final int WRONG_THREAD = 7

The Coordination cannot be ended by the calling thread since the Coordination is on the thread local Coordination stack of another thread.

public CoordinationException(String message, Coordination coordination, int type, Throwable cause)

message The detail message for this exception.

coordination The Coordination associated with this exception.

cause The cause associated with this exception.

type The type of this exception.

Create a new Coordination Exception with a cause.

Throws IllegalArgumentException – If the specified type is FAILED and the specified cause is null.
public CoordinationException(String message, Coordination coordination, int type)

- `message` The detail message for this exception.
- `coordination` The Coordination associated with this exception.
- `type` The type of this exception.

Create a new Coordination Exception.

**Throws** IllegalArgumentException – If the specified type is FAILED.

public long getId()

- Returns the id of the Coordination associated with this exception.

**Returns**

The id of the Coordination associated with this exception or -1 if no Coordination is associated with this exception.

public String getName()

- Returns the name of the Coordination associated with this exception.

**Returns**

The name of the Coordination associated with this exception or "<" if no Coordination is associated with this exception.

public int getType()

- Returns the type for this exception.

**Returns**

The type of this exception.

public final class CoordinationPermission extends BasicPermission

A bundle's authority to create or use a Coordination.

CoordinationPermission has three actions: initiate, participate and admin.

**Concurrency** Thread-safe

public static final String ADMIN = "admin"

The action string admin.

public static final String INITIATE = "initiate"

The action string initiate.

public static final String PARTICIPATE = "participate"

The action string participate.

public CoordinationPermission(String filter, String actions)

- `filter` A filter expression. Filter attribute names are processed in a case sensitive manner. A special value of "*" can be used to match all coordinations.
- `actions` admin, initiate or participate (canonical order).

Creates a new granted CoordinationPermission object. This constructor must only be used to create a permission that is going to be checked.

Examples:

(coordination.name=com.acme.*)
(&(<signer=\*,o=ACME,c=US>(coordination.name=com.acme.*)))
(<signer=\*,o=ACME,c=US>)
When a signer key is used within the filter expression the signer value must escape the special filter chars (‘*’, ‘(‘, ‘)’).

The name is specified as a filter expression. The filter gives access to the following attributes:

- signer - A Distinguished Name chain used to sign the exporting bundle. Wildcards in a DN are not matched according to the filter string rules, but according to the rules defined for a DN chain.
- location - The location of the exporting bundle.
- id - The bundle ID of the exporting bundle.
- name - The symbolic name of the exporting bundle.
- coordination.name - The name of the requested coordination.

Filter attribute names are processed in a case sensitive manner.

Throws IllegalArgumentException – If the filter has an invalid syntax.

130.5.4.5 public CoordinationPermission(String coordinationName,Bundle coordinationBundle,String actions)

coordinationName The name of the requested Coordination.

coordinationBundle The bundle which created the requested Coordination.

actions admin, initiate or participate (canonical order).

Determines a new requested CoordinationPermission object to be used by the code that must perform checkPermission. CoordinationPermission objects created with this constructor cannot be added to an CoordinationPermission permission collection.

130.5.4.6 public boolean equals(Object obj)

obj The object to test for equality with this CoordinationPermission object.

Determines the equality of two CoordinationPermission objects. This method checks that specified permission has the same name and CoordinationPermission actions as this CoordinationPermission object.

Returns true if obj is a CoordinationPermission, and has the same name and actions as this CoordinationPermission object; false otherwise.

130.5.4.7 public String getActions()

Returns the canonical string representation of the CoordinationPermission actions.

Always returns present CoordinationPermission actions in the following order: admin, initiate, participate.

Returns Canonical string representation of the CoordinationPermission actions.

130.5.4.8 public int hashCode()

Returns the hash code value for this object.

Returns A hash code value for this object.

130.5.4.9 public boolean implies(Permission p)

p The requested permission.

Determines if the specified permission is implied by this object.

This method checks that the filter of the target is implied by the coordination name of this object. The list of CoordinationPermission actions must either match or allow for the list of the target object to imply the target CoordinationPermission action.
Returns true if the specified permission is implied by this object; false otherwise.

130.5.4.10 public PermissionCollection newPermissionCollection()

- Returns a new PermissionCollection object suitable for storing CoordinationPermission objects.

Returns A new PermissionCollection object.

130.5.5 public interface Coordinator

A Coordinator service coordinates activities between different parties.

A bundle can use the Coordinator service to create Coordination objects. Once a Coordination object is created, it can be pushed on the thread local Coordination stack to be an implicit parameter as the current Coordination for calls to other parties, or it can be passed directly to other parties as an argument. The current Coordination, which is on the top of the current thread’s thread local Coordination stack, can be obtained with peek().

Any active Coordinations created by a bundle must be terminated when the bundle releases the Coordinator service. The Coordinator service must fail these Coordinations with the RELEASED exception.

A Participant can register to participate in a Coordination and receive notification of the termination of the Coordination.

The following example code shows an example usage of the Coordinator service.

```java
void foo() {
    Coordination c = coordinator.begin("work", 0);
    try {
        doWork();
    } catch (Exception e) {
        c.fail(e);
    } finally {
        c.end();
    }
}
```

In the `doWork` method, code can be called that requires notification of the termination of the Coordination. The `doWork` method can then register a Participant with the Coordination.

```java
void doWork() {
    if (coordinator.addParticipant(this)) {
        beginWork();
    } else {
        beginWork();
        finishWork();
    }
}
```

```java
void ended(Coordination c) {
    finishWork();
}
```

```java
void failed(Coordination c) {
    undoWork();
}
```

Concurrency Thread-safe
Consumer Type Consumers of this API must not implement this type

130.5.5.1 public boolean addParticipant(Participant participant)

Register a Participant with the current Coordination. If there is no current Coordination, this method does nothing and returns false. Otherwise, this method calls Coordination.addParticipant(Participant) with the specified Participant on the current Coordination and returns true.

Returns false if there was no current Coordination, otherwise returns true.

Throws CoordinationException – If the Participant could not be registered with the current Coordination. This exception should normally not be caught by the caller but allowed to be caught by the initiator of this Coordination.

SecurityException – If the caller does not have CoordinationPermission[PARTICIPATE] for the current Coordination.

See Also Coordination.addParticipant(Participant)

130.5.5.2 public Coordination begin(String name,long timeMillis)

Create a new Coordination and make it the current Coordination. This method does that same thing as calling create(name, timeMillis).push()

Returns A new Coordination object

Throws IllegalArgumentException – If the specified name does not follow the symbolic-name syntax or the specified time is negative.

SecurityException – If the caller does not have CoordinationPermission[INITIATE] for the specified name and creating bundle.

130.5.5.3 public Coordination create(String name,long timeMillis)

Create a new Coordination.

Returns The new Coordination object.

Throws IllegalArgumentException – If the specified name does not follow the symbolic-name syntax or the specified time is negative.

SecurityException – If the caller does not have CoordinationPermission[INITIATE] for the specified name and creating bundle.

130.5.5.4 public boolean fail(Throwable cause)

Terminate the current Coordination as a failure with the specified failure cause.

cause The failure cause. The failure cause must not be null.
If there is no current Coordination, this method does nothing and returns \texttt{false}.

Otherwise, this method returns the result from calling \texttt{Coordination.fail(\text{Throwable})} with the specified failure cause on the current Coordination.

\textbf{Returns} \texttt{false} if there was no current Coordination, otherwise returns the result from calling \texttt{Coordination.fail(\text{Throwable})} on the current Coordination.

\textbf{Throws} \texttt{SecurityException} – If the caller does not have \texttt{CoordinationPermission[PARTICIPATE]} for the current Coordination.

\textbf{See Also} \texttt{Coordination.fail(\text{Throwable})}

### 130.5.5.5 \texttt{public Coordination getCoordination(long id)}

\texttt{id} The id of the requested Coordination.

\textbf{Returns} A Coordination having with specified id or \texttt{null} if no Coordination with the specified id exists, the Coordination with the specified id is terminated or the caller does not have \texttt{CoordinationPermission[ADMIN]} for the Coordination with the specified id.

### 130.5.5.6 \texttt{public Collection<Coordination> getCoordinations()}

\textbf{Returns} A snapshot of all active Coordinations. If there are no active Coordinations, the returned list will be empty. The returned collection is the property of the caller and can be modified by the caller.

### 130.5.5.7 \texttt{public Coordination peek()}

\textbf{Returns} The current Coordination or \texttt{null} if the thread local Coordination stack is empty.

### 130.5.5.8 \texttt{public Coordination pop()}

\textbf{Throws} \texttt{SecurityException} – If the caller does not have \texttt{CoordinationPermission[INITIATE]} for the current Coordination.

### 130.5.6 \texttt{public interface Participant}

A Participant participates in a Coordination.
A Participant can participate in a Coordination by registering itself with the Coordination. After successfully registering itself, the Participant is notified when the Coordination is terminated.

If a Coordination terminates normally, then all registered Participants are notified on their ended(Coordination) method. If the Coordination terminates as a failure, then all registered Participants are notified on their failed(Coordination) method.

Participants are required to be thread safe as notification can be made on any thread.

A Participant can only be registered with a single active Coordination at a time. If a Participant is already registered with an active Coordination, attempts to register the Participation with another active Coordination will block until the Coordination the Participant is registered with terminates. Notice that in edge cases the notification to the Participant that the Coordination has terminated can happen before the registration method returns.

**Concurrence** Thread safe

130.5.6.1 **public void ended(Coordination coordination) throws Exception**

coordination The Coordination that has terminated normally.

- Notification that a Coordination has terminated normally.

This Participant should finalize any work associated with the specified Coordination.

**Throws** Exception – If this Participant throws an exception, the Coordinator service should log the exception. The Coordination.end() method which is notifying this Participant must continue notification of other registered Participants. When this is completed, the Coordination.end() method must throw a CoordinationException of type CoordinationException.PARTIALLY_ENDED.

130.5.6.2 **public void failed(Coordination coordination) throws Exception**

coordination The Coordination that has terminated as a failure.

- Notification that a Coordination has terminated as a failure.

This Participant should discard any work associated with the specified Coordination.

**Throws** Exception – If this Participant throws an exception, the Coordinator service should log the exception. The Coordination.fail(Throwable) method which is notifying this Participant must continue notification of other registered Participants.

130.6 **References**

[1] OSGi Core Release 6

http://www.osgi.org/Specifications/HomePage
132 Repository Service Specification

Version 1.1

132.1 Introduction

The guiding force behind the OSGi Specifications is a reusable component model. The OSGi Core Release 6 provides a solid foundation for such a component model by providing a component collaboration framework with a comprehensive management model. The service specifications provide the abstract APIs to allow many different collaborations between components. This Repository Service Specification provides the capability to manage the external access to components and other resources.

Though the Repository service can be used as a standalone service to search and retrieve general binary artifacts, called resources, it is intended to be used in conjunction with the Resolver Service Specification on page 639.

The model of the Repository is based on the generic Requirement-Capability model defined in [3] Resource API Specification, this chapter relies on the definitions of the generic model.

132.1.1 Essentials

- **External** - Provide access to external components and resources.
- **Resolve** - The Repository API must be closely aligned with the Resolver API since they are intended to be used in conjunction.
- **Searching** - Support general queries.
- **Metadata** - Allow resources to provide content information.
- **Retrieval** - Allow the retrieval of Resources from remote locations.
- **Batching** - Repositories must be able to batch queries.
- **Distribution** - Allow Repositories to be defined with a simple storage scheme such that Repositories can be distributed on a removable media like a CD/DVD.
- **Mirroring** - Repositories must be able to support selecting a remote site based on the local situation.

132.1.2 Entities

- **Repository** - A facade to a (remote) set of resources described by capabilities.
- **Resource** - An artifact that has requirements that must be satisfied before it is available but provides capabilities when it becomes available.
- **Requirement** - An expression that asserts a capability.
- **Capability** - Describes a feature of the resource so that it can be required by a requirement.
- **Resource Content** - Provides access to the underlying bytes of the resource in the default format.
132.1.3 Synopsis

There are many different repositories available on the Internet or on fixed media. A repository can be made available to bundles by providing a Repository service. If such a bundle, for example a Management Agent performing a provisioning operation, finds that it has an unmatched requirement then it can query the repository services to find matching capabilities. The Repository service can implement the query in many different ways. It can ship the requirement to a remote side to be processed or it can process the query locally.

This specification also provides an XML schema that can be used to describe a Repository. Instances of this schema can be downloaded from a remote repository for local indexing or they can be stored for example on a DVD together with the resources.

132.2 Using a Repository

The Repository service provides an abstraction to a, potentially remote, set of resources. In the generic Capability-Requirement model, resources are modeled to declare capabilities and requirements. The primary purpose of a Repository is to enable a management agent that uses the Resolver API to leverage a wide array of repositories. This Repository service specification allows different Repository providers to be installed as bundles, and each bundle can register multiple Repository services. The Repository is sufficiently abstract to allow many different implementations.

Repository services are identified by a number of service properties:

- service.pid - A mandatory unique identity for this Repository service.
- service.description - An optional human readable name for this Repository.
- repository.url - Optional URLs to landing pages of the repository, if they exist.

In general, the users of the Repository service should aggregate all services in the service registry. This strategy allows the deployer to control the available Repositories. The following example, using Declarative Service annotations to show the dependencies on the service registry, shows how to aggregate the different Repository services.

List<Repository> repos = new CopyOnWriteArrayList<Repository>();
@Reference(
cardinality = ReferenceCardinality.MULTIPLE,
policy = ReferencePolicy.DYNAMIC)
void addRepository( Repository repo ) { repos.add(repo); }
void removeRepository( Repository repo ) { repos.remove(repo); }

To access a resource in a Repository service it is necessary to construct a requirement, pass this to
the Repository service, and then use the returned capabilities to satisfy the resolver or to get the re-
source from the capability. The Repository then returns all matching capabilities. The requirement
matches the capability if their namespaces match and the requirement's filter is absent or matches
the attributes.

The findProviders(Collection) method takes a Collection of requirements. The reason for this col-
lection is that it allows the caller to specify multiple requirements simultaneously so that Reposito-
ries can batch requests, the requirements in this collection are further unrelated. That is, they do not
form an expression in any way. Multiple requirements as the parameter means that the result must
be a map so that the caller can find out what requirement matched what capabilities. For example:

List<Capability> find( Requirement r ){
    List<Capability> result = new ArrayList<Capability>();

    for ( Repository repo : repos ) {
        Map<Requirement,Collection<Capability>> answer =
            repo.findProviders( Collections.singleton( r ) );
        result.addAll( answer.get( r ) );
    }

    return result;
}

Access to resources is indirect since the Repository returns capabilities. Each capability is declared
in a resource and the getResource() method provides access to the underlying resource. Since each
resource declares an osgi.identity capability it is possible to retrieve a resource from a repository if
the identity name, type, and version are known. For example, to get a bundle resource:

Resource getResource( String type, String name, Version version ) {
    String filter = String.format(
        "(&(type=%s)(osgi.identity=%s)(version=%s))",
        type,
        name,
        version );

    RequirementBuilder builder = repo.newRequirementBuilder("osgi.identity");
    builder.addDirective("filter", filter);
    Requirement r = builder.build();

    List<Capability> capabilities = find( r );
    if ( capabilities.isEmpty() )
        return null;
    return capabilities.get( 0 ).getResource();
}

Resources that originate from Repository services must implement the RepositoryContent interface,
this interface provides stream access to the default storage format. It is therefore possible to get the
content with the following code.

InputStream getContent( String type, String name, Version version ) {
Resource r = getResource( type, name, version );
if ( r == null )
    return null;
return ((RepositoryContent)r).getContent();
}

The getContent() method returns an InputStream in the default format for that resource type. Resources from a Repository should also have one or more osgi.content capabilities that advertise the same resource in the same or different formats. The osgi.content capability has a number of attributes that provide information about the resource's download format:

- osgi.content - A unique SHA-256 for the content as read from the URL.
- url - A URL to the content.
- mime - An IANA MIME type for the content.
- size - Size in bytes of the content.

It is therefore possible to search for a specific MIME type and download that format. For example:

String getURL( String type, String name, Version version, String mime ) throws Exception {
    Resource r = getResource( type, name, version );
    for ( Capability cap : r.getCapabilities( "osgi.content") ) {
        Map<String, Object> attrs = cap.getAttributes();
        String actual = (String) attrs.get( "mime" );
        if ( actual != null && mime.equalsIgnoreCase( actual) ) {
            String url = (String) attrs.get( "url" );
            if ( url != null )
                return url;
        }
    }
    return null;
}

Since the osgi.content capability contains the SHA-256 digest as the osgi.content attribute it is possible to verify the download that it was correct.

Every resource has an osgi.identity capability. This namespace defines, in [2] Framework Namespaces, the possibility to add related resources, for example javadoc or sources. A resource then has informational requirements to osgi.identity capabilities; these requirements are marked with a classifier directive that holds the type of relation. The following example shows how it would be possible to find such a related resource:

InputStream getRelated(Resource resource, String classifier) throws Exception {
    for ( Requirement r : resource.getRequirements( "osgi.identity") ) {
        if ( classifier.equals( r.getDirectives().get( "classifier") ) ) {
            Collection<Capability> capabilities = repository.findProviders( Collections.singleton( r ) ).get( r );
            if ( capabilities.isEmpty() )
                continue;

            Capability c = capabilities.iterator().next();
            Resource related = c.getResource();
            return ((RepositoryContent)related).getContent();
        }
    }
    return null;
}
132.2.1 Combining Requirements

In some cases it may be useful to find resources in the repository that satisfy criteria across multiple namespaces.

A simple Requirement object can contain a filter that makes assertions about capability attributes within a single namespace. So for example, a single requirement can state that a package org.example.mypkg must be exported in a version between 3.1 inclusive and 4.0 exclusive:

```java
RequirementBuilder rb = repo.newRequirementBuilder("osgi.wiring.package");
String rf = "(&(osgi.wiring.package=org.example.mypkg) \\
  +(version>=3.1)(!(version>=4.0)))";
rb.addDirective("filter", rf);
Requirement r = rb.build();
```

This requirement contains three conditions on the osgi.wiring.package capability.

In some situations it may be needed to specify requirements that cover multiple namespaces. For example a bundle might be needed that exports the above package, but the bundle must also have the ASL2 license. A resource's license is available as an attribute on the osgi.identity namespace. Constructing a constraint that combines requirements from multiple namespaces can be done by using an Expression Combiner, which can be obtained from the Repository service. The Repository service provides a findProviders(RequirementExpression) overload that can take a requirement expression and returns a Promise to a collection of matching resources.

```java
RequirementBuilder lb = repo.newRequirementBuilder("osgi.identity");
String lf = "((license=http://opensource.org/licenses/Apache-2.0)";
lb.addDirective("filter", lf);

RequirementExpression expr = repo.getExpressionCombiner().and(
  lb.buildExpression(), rb.buildExpression());

Promise<Collection<Resource>> p = repo.findProviders(expr);
```

// Let findProviders() do its work async and update a ui component
// once the result is available
p.then(new Success<Collection<Resource>, Void>() {
  public void call(Promise<Collection<Resource>> resolved)
    throws Exception {
    ui.update(resolved.getValue());
    return null;
  }
});

// Instead of the async chain above its also possibllye to
// wait for the promise value synchronously:
// Collection<Resource> resources = p.getValue();

For more details on OSGi Promises, see the Promises Specification on page 699.
132.3 Repository

A Repository service provides access to capabilities that satisfy a given requirement. A Repository can be the facade of a remote server containing a large amount of resources, a repository on removable media, or even a collection of bundles inside a ZIP file. A Repository communicates in terms of requirements and capabilities as defined in [3] Resource API Specification. This model is closely aligned with the Resolver Service Specification on page 639.

A Repository service must be registered with the service properties given in the following table.

### Table 132.1 Repository Service Properties

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Opt</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service.pid</td>
<td>mandatory</td>
<td>String</td>
<td>A globally unique identifier for this Repository.</td>
</tr>
<tr>
<td>service.description</td>
<td>optional</td>
<td>String</td>
<td>The Repository Name</td>
</tr>
<tr>
<td>repository.url</td>
<td>optional</td>
<td>String+</td>
<td>URLs related to this Repository.</td>
</tr>
</tbody>
</table>

The Repository implements the following methods:

- **findProviders(Collection)** - For each requirement find all the capabilities that match that requirement and return them as a Map<Requirement,Collection<Capability>>.
- **findProviders(RequirementExpression)** - Find all resources that match the requirement expression. The requirement expression is used to combine multiple requirements using the and, or and not operators.
- **getExpressionCombiner()** - Obtain an expression combiner. This expression combiner is used to produce requirement expressions from simple requirements or other requirement expressions.
- **newRequirementBuilder(String)** - Obtain a convenience builder for Requirement objects.

A Repository must not perform any namespace specific actions or matching. The Repository must therefore match a requirement to a capability with the following rules:

- The namespace must be identical, and
- The requirement's filter is absent or it must match the capability’s attributes.

Resources originating from a Repository service must additionally:

- Implement the RepositoryContent interfaces, see Repository Content on page 536.
- Provide at least one osgi.content Capability, see osgi.content Namespace on page 536.

132.3.1 Repository Content

Resources originating from a Repository must implement the RepositoryContent interface. The purpose of this interface is to allow users of the Repositories access to an Input Stream that provides access to the resource.

The RepositoryContent interface provides a single method:

- **getContent()** - Return an Input Stream for the resource, if more than one osgi.content capability is present the content associated with the first capability is returned.

132.4 osgi.content Namespace

A resource is a logical concept, to install a resource in an environment it is necessary to get access to its contents. A resource can be formatted in different ways. It is possible to deliver a bundle as a JAR file, a Pack200 file, or some other format. In general, the RepositoryContent interface provides access to the default format.
The Repository can advertise the different formats with osgi.content capabilities. Each of those capabilities is identified with a unique SHA-256 checksum and has a URL for the resource in the specified format. The size and mime attributes provide information the download format, this can be used for selection. If more than one osgi.content capability is associated with a resource, the first capability must represent the default format. If the resource has a standard or widely used format (e.g., JAR for bundles and ESA for subsystems), and that format is provided as part of the repository, then that format should be the default format.

The osgi.content Namespace supports the attributes defined in the following table and Content-Namespace.

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>M/O</th>
<th>Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>osgi.content</td>
<td>CA</td>
<td>M</td>
<td>String</td>
<td>[0-9a-fA-F]{64}</td>
<td>The SHA-256 hex encoded digest for this resource</td>
</tr>
<tr>
<td>url</td>
<td>CA</td>
<td>M</td>
<td>String</td>
<td>&lt;url&gt;</td>
<td>The URL to the bytes. This must be an absolute URL.</td>
</tr>
<tr>
<td>size</td>
<td>CA</td>
<td>M</td>
<td>Long</td>
<td>[0-9]+</td>
<td>The size of the resource in bytes as it will be read from the URL.</td>
</tr>
<tr>
<td>mime</td>
<td>CA</td>
<td>M</td>
<td>String</td>
<td>&lt;mime type&gt;</td>
<td>An IANA defined MIME type for the format of this content.</td>
</tr>
</tbody>
</table>

### 132.5 XML Repository Format

This is an optional part of the specification since the Repository interface does not provide access how the Repository obtains its information. However, the purpose of this part of the specification is to provide a commonly recognized format for interchanging Repository metadata.

This section therefore describes an XML schema to represent Repository content. It is expected that Internet based Repositories can provide such an XML file to clients. A Repository XML file can be used as a common interchange format between multiple Repository implementations.

The Repository XML describes a number of resources with their capabilities and requirements. Additionally the XML can refer to other Repository XML files. The XML Schema can be found at its XML namespace, see XML Repository Schema on page 540. The XML structure, which closely follows the Requirement-Capability model, is depicted in Figure 132.2.

![XML Structure Diagram](image-url)
The different elements are discussed in the following sections. All types are derived from the XML Schema types, see [4] XML Schema Part 2: Data types Second Edition.

### 132.5.1 Repository Element

The repository element is the root of the document. The repository element has the following child elements:

- referral*: Referrals to other repositories for a federated model, see Referral Element on page 538.
- resource*: Resource definitions, see Resource Element on page 538.

The repository element has the attributes defined in the following table.

Table 132.3 repository element attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>NCName</td>
<td>The name of this Repository. For informational purposes.</td>
</tr>
<tr>
<td>increment</td>
<td>long</td>
<td>Counter which increments every time the repository is changed. Can be used by clients to check for changes. The counter is not required to increase monotonically.</td>
</tr>
</tbody>
</table>

### 132.5.2 Referral Element

The purpose of the referral element is to allow a Repository to refer to other Repositories, allowing for federated Repositories. Referrals are applied recursively. However, this is not always desired. It is therefore possible to limit the depth of referrals. If the depth attribute is \(\geq 1\), the referred repository must be included but it must not follow any referrals from the referred repository. If the depth attribute is more than one, referrals must be included up to the given depth. Depths of referred repositories must also be obeyed, where referred repositories may reduce the effective depth but not increase it. For example if a top repository specifies a depth of 5 and a level 3 repository has a depth of 1 then the repository on level 5 must not be used. If not specified then there is no limit to the depth. Referrals that have cycles must be ignored, a resource of a given Repository must only occur once in a Repository.

The referral element has the attributes defined in the following table.

Table 132.4 referral element attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>depth</td>
<td>int</td>
<td>The max depth of referrals</td>
</tr>
<tr>
<td>url</td>
<td>anyURI</td>
<td>A URL to where the referred repository XML can be found. The URL can be absolute or relative to the URI of the current XML resource.</td>
</tr>
</tbody>
</table>

### 132.5.3 Resource Element

The resource element defines a Resource. The resource element has the following child elements:

- requirement*: The requirements of this resource, see Requirement Element on page 539.
- capability*: The capabilities of this resource, see Capability Element on page 538.

The Resource element has no attributes.

### 132.5.4 Capability Element

The capability element maps to a capability, it holds the attributes and directives. The capability element has the following child elements:

- directive*: The directives for the capability, see Directive Element on page 540.
• attribute*: The attributes for the capability, see Attribute Element on page 539.

The capability element has the attributes defined in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>token</td>
<td>The namespace of this capability</td>
</tr>
</tbody>
</table>

### 132.5.5 Requirement Element

The requirement element maps to a requirement, it holds the attributes and directives. The requirement element has the following child elements:

• directive*: The directives for the requirement, see Directive Element on page 540.
• attribute*: The attributes for the requirement, see Attribute Element on page 539.

The requirement element has the attributes defined in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>token</td>
<td>The namespace of this requirement</td>
</tr>
</tbody>
</table>

### 132.5.6 Attribute Element

An attribute element describes an attribute of a capability or requirement. Attributes are used to convey information about the Capability-Requirement. Attributes for the capability are used for matching the requirement’s filter. The meaning of attributes is described with the documentation of the namespace in which they reside.

Attributes are optionally typed according to the [1] Framework Module Layer specification. The default type is String, the value of the value attribute. However, if a type attribute is specified and it is not String then the value attribute must be converted according to the type attribute specifier. The syntax of the type attribute is as follows:

- `type ::= list | scalar`
- `list ::= 'List<' scalar '>'`  // no spaces between terminals
- `scalar ::= 'String' | 'Version' | 'Long' | 'Double'`

A list conversion requires the value to be broken in tokens separated by comma (`,`). Whitespace around the list and around commas must be trimmed for non-String types. Each token must then be converted to the given type according to the scalar type specifier. The exact rules for the comma separated lists are defined in [1] Framework Module Layer, see Bundle Capability Attributes.

The conversion of value s, when scalar, must take place with the following methods:

- String - No conversion, use s
- Version - Version.parseVersion(s)
- Long - After trimming whitespace, Long.parseLong(s)
- Double - After trimming whitespace, Double.parseDouble(s)

The attribute element has the attributes defined in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>token</td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>The value of the attribute.</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td>The type of the attribute, the syntax is outlined in the previous paragraphs.</td>
</tr>
</tbody>
</table>
132.5.7 Directive Element

A directive element describes a directive of a capability or a requirement. Directives are used to convey information about the Capability-Requirement. The meaning of directives is described with the documentation of the namespace in which they reside.

The directive element has the attributes defined in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>token</td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>The value of the attribute</td>
</tr>
</tbody>
</table>

132.5.8 Sample XML File

The following example shows a very small XML file. The file contains one resource.

```xml
<repository name='OSGiRepository' increment='13582741' xmlns='http://www.osgi.org/xmlns/repository/v1.0.0'>
  <resource>
    <requirement namespace='osgi.wiring.package'>
      <directive name='filter' value='(&(osgi.wiring.package=org.apache.commons.pool)(version&gt;=1.5.6))'/>
    </requirement>
    <requirement namespace='osgi.identity'>
      <directive name='effective' value='meta'/>
      <directive name='resolution' value='optional'/>
      <directive name='filter' value='(&(version=1.5.6)(osgi.identity=org.acme.pool-src))'/>
      <directive name='classifier' value='sources'/>
    </requirement>
    <capability namespace='osgi.identity'>
      <attribute name='osgi.identity' value='org.acme.pool'/>  
      <attribute name='version'type='Version' value='1.5.6'/>  
      <attribute name='type' value='osgi.bundle'/>  
    </capability>
    <capability namespace='osgi.content'>
      <attribute name='osgi.content' value='e3b0c44298fc1c149afbf4c8996fb92427a41e4649b934ca495991b7852b855'/>
      <attribute name='url' value='http://www.acme.com/repository/org/acme/pool/org.acme.pool-1.5.6.jar'/>
      <attribute name='size' type='Long' value='4405'/>  
      <attribute name='mime' value='application/vnd.osgi.bundle'/>  
    </capability>
    <capability namespace='osgi.wiring.bundle'>
      <attribute name='osgi.wiring.bundle' value='org.acme.pool'/>  
      <attribute name='bundle-version' type='Version' value='1.5.6'/>  
    </capability>
    <capability namespace='osgi.wiring.package'>
      <attribute name='osgi.wiring.package' value='org.acme.pool'/>  
      <attribute name='version'type='Version' value='1.1.2'/>  
      <attribute name='bundle-version' type='Version' value='1.5.6'/>  
      <attribute name='bundle-symbolic-name' value='org.acme.pool'/>  
      <directive name='uses' value='org.acme.pool.org.acme.util'/>  
    </capability>
  </resource>
</repository>
```

132.6 XML Repository Schema

The namespace of this schema is:
The schema for this namespace can be found at the location implied in its name. The recommended prefix for this namespace is repo.

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
       xmlns:repo="http://www.osgi.org/xmlns/repository/v1.0.0"
       targetNamespace="http://www.osgi.org/xmlns/repository/v1.0.0"
       elementFormDefault="unqualified"
       attributeFormDefault="unqualified"
       version="1.0.1">
  <element name="repository" type="repo:Trepository" />
  <complexType name="Trepository">
    <sequence>
      <choice minOccurs="0" maxOccurs="unbounded">
        <element name="resource" type="repo:Tresource" />
        <element name="referral" type="repo:Treferral" />
      </choice>
      <!-- It is non-deterministic, per W3C XML Schema 1.0: http://www.w3.org/TR/xmlschema-1/#cos-nonambig to use namespace="##any" below. -->
      <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
    <attribute name="name" type="string">
      <annotation>
        <documentation xml:lang="en">
          The name of the repository. The name may contain spaces and punctuation.
        </documentation>
      </annotation>
    </attribute>
    <attribute name="increment" type="long">
      <annotation>
        <documentation xml:lang="en">
          An indication of when the repository was last changed. Client's can check if a repository has been updated by checking this increment value.
        </documentation>
      </annotation>
    </attribute>
    <anyAttribute processContents="lax" />
  </complexType>

  <complexType name="Tresource">
    <annotation>
      <documentation xml:lang="en">
        Describes a general resource with requirements and capabilities.
      </documentation>
    </annotation>
    <sequence>
      <element name="requirement" type="repo:Trequirement" minOccurs="0" maxOccurs="unbounded" />
      <element name="capability" type="repo:Tcapability" minOccurs="1" maxOccurs="unbounded" />
      <!-- It is non-deterministic, per W3C XML Schema 1.0: http://www.w3.org/TR/xmlschema-1/#cos-nonambig to use namespace="##any" below. -->
      <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
    <anyAttribute processContents="lax" />
  </complexType>

  <complexType name="Treferral">
    <annotation>
      <documentation xml:lang="en">
        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>
      </annotation>
    </attribute>
  </complexType>
</schema>
```
The depth of referrals this repository acknowledges.

The URL to the referred repository. The URL can be absolute or relative from the given repository's URL.
<annotation>
  <documentation xml:lang="en">
    A named value with an optional type that decorates a requirement or capability.
  </documentation>
</annotation>

<sequence>
  <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
</sequence>

<attribute name="name" type="string">
  <annotation>
    <documentation xml:lang="en">
      The name of the attribute.
    </documentation>
  </annotation>
</attribute>

<attribute name="value" type="string">
  <annotation>
    <documentation xml:lang="en">
      The value of the attribute.
    </documentation>
  </annotation>
</attribute>

<attribute name="type" type="repo:TpropertyType" default="String">
  <annotation>
    <documentation xml:lang="en">
      The type of the attribute.
    </documentation>
  </annotation>
</attribute>

<attribute processContents="lax" />
</complexType>

<complexType name="Tdirective">
  <annotation>
    <documentation xml:lang="en">
      A named value of type string that instructs a resolver how to process a requirement or capability.
    </documentation>
  </annotation>
  <sequence>
    <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
  </sequence>

  <attribute name="name" type="string">
    <annotation>
      <documentation xml:lang="en">
        The name of the directive.
      </documentation>
    </annotation>
  </attribute>
  
  <attribute name="value" type="string">
    <annotation>
      <documentation xml:lang="en">
        The value of the directive.
      </documentation>
    </annotation>
  </attribute>

  <attribute processContents="lax" />
</complexType>

<simpleType name="TpropertyType">
  <restriction base="string">
    <enumeration value="String" />
    <enumeration value="Version" />
    <enumeration value="Long" />
    <enumeration value="Double" />
    <enumeration value="List&lt;String&gt;" />
    <enumeration value="List&lt;Version&gt;" />
    <enumeration value="List&lt;Long&gt;" />
    <enumeration value="List&lt;Double&gt;" />
  </restriction>
</simpleType>

<attribute name="must-understand" type="boolean" default="false"/>
132.7 Security

132.7.1 External Access

Repositories in general will get their metadata and artifacts from an external source, which makes them an attack vector for a malevolent Bundle that needs unauthorized external access. Since a Bundle using a Repository has no knowledge of what sources the Repository will access it will be necessary for the Repository to implement the external access in a `doPrivileged` block. Implementations must ensure that callers cannot influence/modify the metadata in such a way that the `getContent()` method could provide access to arbitrary Internet resources. This could for example happen if:

- The implementation relies on the osgi.content namespace to hold the URL
- The attributes Map from the osgi.content Capability is modifiable

If the malevolent Bundle could change the osgi.content attribute it could change it to arbitrary URLs. This example should make it clear that Repository implementations must be very careful.

132.7.2 Minimum Implementation Permissions

PackagePermission[org.osgi.service.repository, IMPORT]
ServicePermission[...Repository, REGISTER ]
SocketPermission[ ... carefully restrict external access...]

132.7.3 Minimum Using Permissions

PackagePermission[org.osgi.service.repository, IMPORT ]
ServicePermission[...Repository, GET ]

132.8 org.osgi.service.repository

Repository Service Package Version 1.1.

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.repository; version="[1.1,2.0)"

Example import for providers implementing the API in this package:
Import-Package: org.osgi.service.repository; version="[1.1,1.2)"

132.8.1 Summary

- AndExpression - A RequirementExpression representing the and of a number of requirement expressions.
- ContentNamespace - Content Capability and Requirement Namespace.
- ExpressionCombiner - An ExpressionCombiner can be used to combine requirement expressions into a single complex requirement expression using the and, or and not operators.
- IdentityExpression - A RequirementExpression representing a requirement.
- NotExpression - A RequirementExpression representing the not (negation) of a requirement expression.
- OrExpression - A RequirementExpression representing the or of a number of requirement expressions.
- Repository - A repository service that contains resources.
- RepositoryContent - An accessor for the content of a resource.
- RequirementBuilder - A builder for requirements.
- RequirementExpression - The super interface for all requirement expressions.

132.8.2

public interface AndExpression extends RequirementExpression

A RequirementExpression representing the and of a number of requirement expressions.

Since 1.1

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

132.8.2.1

public List<RequirementExpression> getRequirementExpressions()

Return the requirement expressions that are combined by this AndExpression.

Returns An unmodifiable list of requirement expressions that are combined by this AndExpression. The list contains the requirement expressions in the order they were specified when this requirement expression was created.

132.8.3

public final class ContentNamespace extends Namespace

Content Capability and Requirement Namespace.

This class defines the names for the attributes and directives for this namespace. All unspecified capability attributes are of type String and are used as arbitrary matching attributes for the capability. The values associated with the specified directive and attribute keys are of type String, unless otherwise indicated.

Concurrency Immutable

132.8.3.1

public static final String CAPABILITY_MIME_ATTRIBUTE = "mime"

The capability attribute that defines the IANA MIME Type/Format for this content.

132.8.3.2

public static final String CAPABILITY_SIZE_ATTRIBUTE = "size"

The capability attribute that contains the size, in bytes, of the content. The value of this attribute must be of type Long.

132.8.3.3

public static final String CAPABILITY_URL_ATTRIBUTE = "url"

The capability attribute that contains the URL to the content.

132.8.3.4

public static final String CONTENT_NAMESPACE = "osgi.content"

Namespace name for content capabilities and requirements.
Also, the capability attribute used to specify the unique identifier of the content. This identifier is the SHA-256 hash of the content.

### 132.8.4 public interface ExpressionCombiner

An ExpressionCombiner can be used to combine requirement expressions into a single complex requirement expression using the and, or and not operators.

**Since** 1.1

**Concurrency** Thread-safe

**Provider Type** Consumers of this API must not implement this type

#### 132.8.4.1 public AndExpression and(RequirementExpression expr1, RequirementExpression expr2)

- **expr1** The first requirement expression to combine into the returned requirement expression.
- **expr2** The second requirement expression to combine into the returned requirement expression

**Returns** An AndExpression representing an and of the specified requirement expressions.

#### 132.8.4.2 public AndExpression and(RequirementExpression expr1, RequirementExpression expr2, RequirementExpression ... moreExprs)

- **expr1** The first requirement expression to combine into the returned requirement expression.
- **expr2** The second requirement expression to combine into the returned requirement expression
- **moreExprs** Optional, additional requirement expressions to combine into the returned requirement expression.

**Returns** An AndExpression representing an and of the specified requirement expressions.

#### 132.8.4.3 public IdentityExpression identity(Requirement req)

- **req** The requirement to wrap in a requirement expression.

**Returns** An IdentityExpression representing the specified requirement.

#### 132.8.4.4 public NotExpression not(RequirementExpression expr)

- **expr** The requirement expression to negate.

**Returns** A NotExpression representing the not of the specified requirement expression.

#### 132.8.4.5 public OrExpression or(RequirementExpression expr1, RequirementExpression expr2)

- **expr1** The first requirement expression to combine into the returned requirement expression.
- **expr2** The second requirement expression to combine into the returned requirement expression

**Returns** An OrExpression representing an or of the specified requirement expressions.

#### 132.8.4.6 public OrExpression or(RequirementExpression expr1, RequirementExpression expr2, RequirementExpression ... moreExprs)

- **expr1** The first requirement expression to combine into the returned requirement expression.
- **expr2** The second requirement expression to combine into the returned requirement expression

...
**moreExprs** Optional, additional requirement expressions to combine into the returned requirement expression.

- Combine multiple RequirementExpressions into a requirement expression using the or operator.

**Returns** An OrExpression representing an or of the specified requirement expressions.

### 132.8.5 public interface IdentityExpression extends RequirementExpression

A RequirementExpression representing a requirement.

**Since** 1.1

**Concurrency** Thread-safe

**Provider Type** Consumers of this API must not implement this type

#### 132.8.5.1 public Requirement getRequirement()

- Return the Requirement contained in this IdentityExpression.

**Returns** The requirement contained in this IdentityExpression.

### 132.8.6 public interface NotExpression extends RequirementExpression

A RequirementExpression representing the not (negation) of a requirement expression.

**Since** 1.1

**Concurrency** Thread-safe

**Provider Type** Consumers of this API must not implement this type

#### 132.8.6.1 public RequirementExpression getRequirementExpression()

- Return the requirement expression that is negated by this NotExpression.

**Returns** The requirement expression that is negated by this NotExpression.

### 132.8.7 public interface OrExpression extends RequirementExpression

A RequirementExpression representing the or of a number of requirement expressions.

**Since** 1.1

**Concurrency** Thread-safe

**Provider Type** Consumers of this API must not implement this type

#### 132.8.7.1 public List<RequirementExpression> getRequirementExpressions()

- Return the requirement expressions that are combined by this OrExpression.

**Returns** An unmodifiable list of requirement expressions that are combined by this OrExpression. The list contains the requirement expressions in the order they were specified when this requirement expression was created.

### 132.8.8 public interface Repository

A repository service that contains resources.

Repositories may be registered as services and may be used as by a resolve context during resolver operations.

Repositories registered as services may be filtered using standard service properties.
**Concurrency**  
Thread-safe

**Provider Type**  
Consumers of this API must not implement this type

### 132.8.8.1 public static final String URL = "repository.url"

Service property to provide URLs related to this repository.

The value of this property must be of type String, String[], or Collection<String>.

### 132.8.8.2 public Map<Requirement,Collection<Capability>> findProviders(Collection<? extends Requirement> requirements)

**requirements**  
The requirements for which matching capabilities should be returned. Must not be null.

- Find the capabilities that match the specified requirements.

**Returns**  
A map of matching capabilities for the specified requirements. Each specified requirement must appear as a key in the map. If there are no matching capabilities for a specified requirement, then the value in the map for the specified requirement must be an empty collection. The returned map is the property of the caller and can be modified by the caller. The returned map may be lazily populated, so calling size() may result in a long running operation.

### 132.8.8.3 public Promise<Collection<Resource>> findProviders(RequirementExpression expression)

**expression**  
The RequirementExpression for which matching capabilities should be returned. Must not be null.

- Find the resources that match the specified requirement expression.

**Returns**  
A promise to a collection of matching Resources. If there are no matching resources, an empty collection is returned. The returned collection is the property of the caller and can be modified by the caller. The returned collection may be lazily populated, so calling size() may result in a long running operation.

**Since**  
1.1

### 132.8.8.4 public ExpressionCombiner getExpressionCombiner()

- Return an expression combiner. An expression combiner can be used to combine multiple requirement expressions into more complex requirement expressions using and, or and not operators.

**Returns**  
An ExpressionCombiner.

**Since**  
1.1

### 132.8.8.5 public RequirementBuilder newRequirementBuilder(String namespace)

**namespace**  
The namespace for the requirement to be created.

- Return a new RequirementBuilder which provides a convenient way to create a requirement.

For example:

```java
Requirement myReq = repository.newRequirementBuilder("org.foo.ns1").
    addDirective("filter", "(org.foo.ns1=val1)"),
    addDirective("cardinality", "multiple").build();
```

**Returns**  
A new requirement builder for a requirement in the specified namespace.

**Since**  
1.1

### 132.8.9 public interface RepositoryContent

An accessor for the content of a resource. All Resource objects which represent resources in a Repository must implement this interface. A user of the resource can then cast the Resource object to this type and then obtain an InputStream to the content of the resource.
Concurrency Thread-safe
Provider Type Consumers of this API must not implement this type

132.8.9.1 public InputStream getContent()

- Returns a new input stream to the content of this resource. The content is represented on the resource through the osgi.content capability. If more than one such capability is associated with the resource, the first such capability is returned.

Returns A new input stream for associated content.

132.8.10 public interface RequirementBuilder

A builder for requirements.

Since 1.1
Provider Type Consumers of this API must not implement this type

132.8.10.1 public RequirementBuilder addAttribute(String name, Object value)

- name The attribute name.
- value The attribute value.
- Add an attribute to the set of attributes.

Returns This requirement builder.

132.8.10.2 public RequirementBuilder addDirective(String name, String value)

- name The directive name.
- value The directive value.
- Add a directive to the set of directives.

Returns This requirement builder.

132.8.10.3 public Requirement build()

- Create a requirement based upon the values set in this requirement builder.

Returns A requirement created based upon the values set in this requirement builder.

132.8.10.4 public IdentityExpression buildExpression()

- Create a requirement expression for a requirement based upon the values set in this requirement builder.

Returns A requirement expression created for a requirement based upon the the values set in this requirement builder.

132.8.10.5 public RequirementBuilder setAttributes(Map<String, Object> attributes)

- attributes The map of attributes.
- Replace all attributes with the attributes in the specified map.

Returns This requirement builder.

132.8.10.6 public RequirementBuilder setDirectives(Map<String, String> directives)

- directives The map of directives.
- Replace all directives with the directives in the specified map.

Returns This requirement builder.
public RequirementBuilder setResource(Resource resource)

resource The resource.

Set the Resource.

A resource is optional. This method will replace any previously set resource.

Returns This requirement builder.

public interface RequirementExpression

The super interface for all requirement expressions. All requirement expressions must extend this interface.

Since 1.1

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

References

[1] Framework Module Layer
   OSGi Core, Chapter 3 Module Layer

   OSGi Core, Chapter 8, osgi.identity Namespace

   OSGi Core, Chapter 6 Resource API Specification

   http://www.w3.org/TR/xmlschema-2/

Changes

- Introduction of Requirement Expressions to represent combined requirements.
- Introduction of Requirement Builder to facilitate the creation of Requirement objects.
- Changes to default content for the Repository Content API and osgi.content capability.
133 Service Loader Mediator Specification

Version 1.0

133.1 Introduction

Java SE 6 introduced the Service Loader, a simple service-provider loading facility, that attempted to unify the different ad-hoc mechanisms used by Java's many factories and builders. The design allows a JAR to advertise the name of one or more embedded classes that implement a given interface and consumers to obtain instances of these implementation classes through the Service Loader API.

Though the Service Loader is about extensibility, its own design is closed and therefore not extendable. It does not support a provider model that would allow different ways of finding interface implementations; its classes are final and its policy is fixed. Unfortunately, the Service Loader's fixed design uses a non-modular class loading policy; it defines its visibility scope with a class loader, which in general requires full visibility of the application's class path. The Service Loader can therefore in OSGi not find implementations from other bundles. Additionally, the Service Loader also does not enforce a life cycle; objects are handed out forever.

Since the Service Loader is the only standardized plugin mechanism in the JRE it is necessary that the mechanism is supported in OSGi with as few changes as possible from the consumer's authors. This specification therefore defines a mediator that ensures that the Service Loader is useful in an OSGi Framework, allowing programs that leverage the Service Loader to be used in OSGi frameworks almost as-is.

133.1.1 Essentials

- **Compatibility** - Allow JARs that run in a classic Java SE environment that leverage the Service Loader to run in OSGi with only manifest modifications.
- **Services** - Register services for Service Provider bundles that opt-in.
- **Security** - Enforce service permissions for the Service Loader objects.
- **Life Cycle** - Manage the life cycle mismatch between OSGi bundles and the Service Loader's create only model.

133.1.2 Entities

- **Service Loader** - An API in Java SE that allows a Consumer to find an implementation of a Service Type from a Service Provider by searching a class loader for Service Providers.
- **Service Type** - The interface or class that the Service Provider must implement/extend.
- **Provider Configuration File** - A resource in the META-INF/services directory that has the fully qualified name of the Service Type and contains one or more fully qualified names of Service Providers.
- **Service Provider** - An implementation class that implements or extends the Service Type.
- **Consumer** - A class that uses the Java SE Service Loader inside an OSGi framework.
- **Mediator** - An extender that mediates between Consumer bundles, the Service Loader API, and Service Provider bundles in an OSGi environment. It consists of a Processor and a Registrar.
• **Processor** - Modifies a bundle that uses the Service Loader API so that it works in an OSGi environment.
• **Registrar** - Registers services on behalf of a bundle that contains Service Providers.

**Figure 133.1  Entities**

- **Consumer**
- **Any OSGi Service user**
- **Processor**
- **Mediator**
- **Registrar**
- **Service Provider**
- **Service Type**
- **osgi.extender= osgi.serviceloader.registrar**
- **osgi.serviceloader**

**133.1.3 Synopsis**

This specification defines two different functions that are provided by a Mediator extender:

- Register OSGi services for each Service Provider.
- Allow Consumers that uses the Service Loader API to access Service Providers from other bundles that would normally not be visible from a bundle.

A Service Provider bundle can provide access to all its Service Providers through OSGi services by declaring a requirement on the `osgi.serviceloader.registrar` extender. This requirement activates a Mediator to inspect the `osgi.serviceloader` capabilities. If no `register` directive is used then all Service Providers for the given Service Type must be registered. Otherwise, each capability can select one Service Provider with the `register` directive. The fully qualified name selects a specific Service Provider, allowing different Service Providers to be registered with different service properties. The Mediator will then register an OSGi service factory for each selected capability. The `osgi.serviceloader` capability's attributes are used to decorate the OSGi service registration with service properties. The service factory returns a new instance for each service get.

Consumers are classes that use the Service Loader API to find Service Provider instances. Since the Service Loader API requires full visibility the Service API fails to work inside an OSGi bundle. A `osgi.serviceloader.processor` extender, which is the Mediator, processes bundles that require this capability by modifying calls to the Service Loader API to ensure that the Service Loader has visibility to published Service Providers.

A Consumer's bundle by default receives visibility to all published Service Providers. Service Providers are published when a bundle declares one or more `osgi.serviceloader` capabilities for a Service Type. If the Consumer has an `osgi.serviceloader` requirement for the given Service Type then the Mediator must only expose the bundles that are wired to those requirements and for each bundle provide all its Service Providers.
Java Service Loader API

Java is quite unique with its focus on separation of specification and implementation. Virtually all Java Specification Requests (JSR) provide a specification that can be implemented independently by different parties. Though this is one of the industry’s best practices it raises a new problem: how to find the implementation available in a Java environment from only the Service Type. A Service Type is usually an interface but a base class can also be used.

Finding a Service Provider (the implementation class) from a Service Type is the so called instance coupling problem. The use of Service Types removed the type coupling between the Consumer of the contract and the Service Provider of the contract (the implementation) but to make things work there is a need of at least one place where the Service Provider is instantiated. The very large number of factories in Java reflects that this is a very common problem.

The general pattern for factories to find Service Providers was to search the class loaders for classes with constant names, varying the package names, often using System properties to extend the different areas to be sought. Though a general pattern based on class loading tricks emerged in the Java VM and application programs, all these patterns differed in details and places where they looked. This became harder and harder to maintain and often caused unexpected instances to be found.

The java.util.ServiceLoader class was therefore first introduced in Java SE 6 to provide a generic solution to this problem, see [1] Java Service Loader API. With this API Service Providers of a specification can now advertise their availability by creating a Provider Configuration File in their JAR in the META-INF/services directory. The name of this resource is the fully qualified name of the Service Type, the Service Provider provides when instantiated.

The Provider Configuration File contains a number of lines with comments or a class name that implements/extends the Service Type. For example:

```
org.example.Foo
```

A Service Provider must then advertise itself like:

```
META-INF/services/org.example.Foo:
  # Foo implementation
  org.acme.impl.FooImplementation
```

The Service Loader API finds all advertisers by constructing the name of the Provider Configuration File from the Service Type and then calling the getResources method on the provided class loader. This returns an enumeration of URLs to the advertisements. It then parses the contents of the resources; that will provide it with a list of Service Providers for the sought Service Type without duplicates. The API will return an iterator that will instantiate an object for the next available Service Provider.

To find the Configuration files for a given Service Type, the Service Loader uses a class loader. The Consumer can select the following class loaders:

- A given class loader as an argument in the call to the constructor
- The Thread Context Class Loader (TCCL)
- The system loader (when null is passed or no TCCL is set)

The class loader restricts the visibility of the Service Loader to only the resources to which the class loader has visibility. If the Service Loader has no access to the advertisement of a Service Provider then it cannot detect it and it will thus not be found.

The Service Provider is loaded from the given class loader, however, the Class.forName method is used, which stores it in the cache of the initiating class loader. This means that Service Providers are
not garbage collected as long as there is a resolved bundle that used the Service Loader to get that Service Provider.

In the Service Loader API, the class does not have to originate from the same JAR file as the advertisement. In OSGi this is more restricted, the advertisement must come from the same bundle or must be explicitly imported.

For example, to load a Foo instance the following code could be used:

```java
ServiceLoader<Foo> sl = ServiceLoader.load( Foo.class );
Iterator<Foo> it = sl.iterator();
if ( it.hasNext() ) {
    Foo foo = it.next();
    ...
}
```

Though the Service Loader API is about extensibility and contract based programming it is in itself not extendable nor replaceable. The `ServiceLoader` class is final, it comes from a sealed JAR, and is in a java package. It also does not provide an API to provide alternate means to find implementations for a Service Type.

### 133.3 Consumers

**Consumers** are classes that are not OSGi aware and directly use the Service Loader API. The Service Loader has a non-modular design and Consumers therefore run into many issues when running in an OSGi framework. Consumers should therefore in general be converted to use the OSGi service layer since this solves the visibility issues, life cycle impedance mismatch, and other problems. The Consumer part of this specification is therefore a last resort to use when existing code uses the Service Loader API and cannot be modified to leverage the OSGi service layer.

#### 133.3.1 Processing

The Service Loader Mediator can *process* the Consumer by modifying calls to the Service Loader API. This specification does not detail how the Mediator ensures that the Consumer has visibility to other Service Providers. However, a Mediator could for example set an appropriate Thread Context Class Loader during the call to the Service Loader's constructor by weaving the Consumer's byte codes.

#### 133.3.2 Opting In

Processing is an opt-in process, the Consumer bundle must declare that it is willing to be processed. The opt-in is handled by a requirement to the `osgi.serviceloader.processor` extender. This requirement must have a single cardinality (the default) since the Mediator uses the wiring to select the Consumer to process when multiple Mediators are present.

For example, the following requirement in a manifest enables a bundle to be processed:

```manifest
Require-Capability:
   .osgi.extender;
    filter:="(&(osgi.extender=osgi.serviceloader.processor)
    (version>=1.0)(!(version>=2.0)))"
```

If the extender `osgi.serviceloader.processor` requirement is satisfied then the wired Mediator must process the Consumer.

The Mediator must give visibility to all bundles with *published* Service Providers unless the Consumer restricts the visibility by having `osgi.serviceloader` requirements. Bundles publish a Service
Type, meaning all their Service Providers for that type, by having at least one osgi.serviceloader capability for that Service Type.

### 133.3.3 Restricting Visibility

A Consumer's bundle can restrict its visibility to certain bundles by declaring an osgi.serviceloader requirement for each Service Type it wants to use. Only bundles wired from those requirement provide their advertised Service Providers. If no such requirements are declared then all bundles with the published Service Type become available.

The cardinality can be used to select a single Service Provider's bundle or multiple bundles if it needs to see all Service Provider bundles. The requirement can be made optional if the Consumer's bundle can work also when no Service Provider bundle is available. See osgi.serviceloader Namespace on page 561 for more details.

For example, a requirement that restricts visibility to the org.example.Foo Service Providers could look like:

```
Require-Capability:
    osgi.serviceloader;
    filter:="(osgi.serviceloader=org.example.Foo)";
    cardinality:="multiple"
```

In this example, any bundle that publishes the org.example.Foo Service Type will contribute its Service Providers.

Visibility can also be restricted to bundles that publish with capability's attributes. Any bundle that has at least one matching capability will then be able to contribute all its Service Providers. For example, the following example selects only bundles that have the classified property set:

```
osgi.serviceloader; filter:="(classified=*)"
```

With Service Registrations, see Registering Services on page 558, the capability can discriminate between multiple Service Providers in the same bundle. The Service Loader API does not have this feature: any wired requirement has visibility to all Service Providers in the wired bundle, regardless of the registered directive.

### 133.3.4 Life Cycle Impedance Mismatch

A Consumer can only see Service Provider instances of bundles that are active during the time the next instance is created. That is, the Mediator must treat the life cycle of the Service Provider as if it was a service. However, the Service Loader implementations perform extensive class loader techniques and cache results. The exact life cycle of the Service Provider bundle with respect to the Consumer is therefore impossible to enforce.

The Service Loader API does not have a life cycle, objects are assumed to stay alive during the duration of the VM's process and due to the use of Class.forName in the Service Loader implementations. Therefore a Mediator should refresh a Consumer bundle when it is using a Service Provider and that Service Provider's bundle becomes stopped otherwise long running applications can run out of memory when bundles are regularly updated.

### 133.3.5 Consumer Example

A legacy JAR for which there is no more source code uses the Service Loader API to get access to com.example.Codec instances through the Service Loader API.

It is wrapped in a bundle that then has the following manifest:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
```
The manifest must then declare that the bundle must be processed, this is triggered by requiring the osgi.serviceloader.processor extender:

Require-Capability:
   osgi.extender;
   filter:="(&(osgi.extender=osgi.serviceloader.processor)
   (version>=1.0)!(version>=2.0))"

With this manifest, the Consumer bundle has full visibility to all Service Provider bundles that are published. The following lines can be added to restrict the visibility to codecs that have support for WAVE formats (although all Service Providers in that bundle will be visible to the consumer).

   osgi.serviceloader;
   filter:="(&(format=WAVE)(osgi.serviceloader=com.example.Codec))"

133.4  Service Provider Bundles

A Service Provider bundle is a bundle that contains one or more Service Providers that are usable by the Service Loader API. This section shows how Service Provider bundles should be constructed and what options they have.

133.4.1 Advertising

Service Providers are implementation classes that are advertised under a Service Type according to the rules in the Service Loader API. A Service Provider is advertised with a Provider Configuration File in a JAR. In an OSGi environment the Service Provider must reside in the same bundle as the advertisement or be imported. A single Provider Configuration File can contain multiple Service Providers. See Java Service Loader API on page 553.

133.4.2 Publishing the Service Providers

Service Providers can be used in two different scenarios:

- A Service Provider can be used by a processed Consumer as a Service Type, or
- It can be registered as a service.

A Service Type must be published to allow its use in these scenarios. Publishing a Service Type consists of providing one or more osgi.serviceloader capabilities for an advertised Service Type, see osgi.serviceloader Namespace on page 561. These osgi.serviceloader capabilities must specify a fully qualified class name of the Service Type, there is no wildcarding allowed. Therefore, publishing a service implicitly makes all corresponding Service Providers available to Consumers.

If a bundle does not provide osgi.serviceloader capabilities then it does not publish any Service Providers and its Service Providers can therefore not be used by Consumers. They can then also not be registered as OSGi services, see OSGi Services on page 557. Tools can use the advertisement of the Service Provider in the JAR to automatically generate the osgi.serviceloader capabilities in the manifest.

For example, the following capability publishes all the Service Providers in its bundle that advertise the com.example.Codec interface:
A Service Provider bundle must not require the osgi.serviceloader.processor extender unless it needs to be processed; publishing a Service Type is sufficient to allow Consumers to use the published Service Types.

133.4.3 OSGi Services

The Service Provider can have its osgi.serviceloader capabilities be registered as services that provide instances from the Service Providers. For this, the Service Provider bundle must require the osgi.serviceloader.registrar extender, which is the Mediator. For example:

Require-Capability:
  osgi.extender;
  filter:="(&(osgi.extender=osgi.serviceloader.registrar)
    (version>=1.0)(!(version>=2.0)))"

The registrar must then inspect each osgi.serviceloader capability and register an associated OSGi Service for each Service Provider selected by that capability. A Service Provider is selected when:

- The capability has no register directive, or
- The register directive matches the fully qualified name of the Service Provider.

A register directive selects a Service Provider if it contains the fully qualified name of the Service Provider, that is, the implementation class. Selection only works for services, Consumer will always see all Service Providers regardless of the register directive due to limitations in the Service Loader API.

For example, the following manifest selects all Service Providers of the com.example.Foo Service Type since no register directive is present:

Provide-Capability:
  osgi.serviceloader;
  uses:="com.example";
  osgi.serviceloader=com.example.Foo

Selected Service Providers must be registered as defined in Registering Services on page 558, with the capability's attributes as decorating service properties. Private service properties (attributes that start with a full stop (".")) and the defined capability attributes in the osgi.serviceloader namespace are not registered as service properties.

The following example would register the format service property but not the .hint service property for the com.acme.impl.WaveFoo Service Provider.

osgi.serviceloader;
  osgi.serviceloader=com.example.Foo;
  uses:="com.example";
  format=WAVE;
  .hint=E5437Qy7;
  register:="com.acme.impl.WaveFoo"

The Mediator must only register OSGi services for selected Service Providers; the Service Provider bundle can therefore decide not to register certain Service Providers and register them with another mechanism, for example Declarative Services or in a bundle activator.
Since the Mediator must use the bundle context of the Service Provider to register the OSGi service, the Service Provider bundle must have the proper Service Permission REGISTER for the Service Type.

### 133.4.4 Service Provider Example

A Foo Codecs JAR needs to be ported to OSGi, it provides a Service Provider for the org.example.Codec Service Type. In this example the JAR is given a new manifest:

```markdown
Manifest-Version:       1.0
Bundle-ManifestVersion: 2
Bundle-SymbolicName:    com.example.foo.codecs
Import-Package:         com.example; version=3.45

To ensure that the bundle opts in to registering its services it must require the osgi.serviceloader.registrar extender.

Require-Capability:
  osgi.extender:
    filter:="(&(osgi.extender=osgi.serviceloader.registrar)
      (version>=1.0)(!(version>=2.0)))"

To publish two Service Providers for the same type, two capabilities must be declared:

Provide-Capability:
  osgi.serviceloader:
    osgi.serviceloader=com.example.Codec;
    format:List<String>="WAVE,WMF";
    register:="com.acme.impl.FooWaveCodec";
    uses:="com.example,org.apache.common.codecs",
  osgi.serviceloader:
    osgi.serviceloader=com.example.Codec;
    format:List<String>=SINUS;
    register:="com.acme.impl.sinus.FooSinusCodec";
    uses:="com.example"

This example implicitly publishes the Service Type com.example.Codec multiple times with different attributes. Consumers that match any of these capabilities will however have visibility to all Service Providers since the Service Loader API cannot discriminate between different Service Providers from the same bundle.

### 133.5 Service Loader Mediator

A Mediator is the osgi.serviceloader.processor and osgi.serviceloader.registrar extender bundle that has the following responsibilities:

- It registers selected Service Providers as OSGi services.
- It processes any Consumers so that Service Loader API calls have proper visibility to published Service Provider bundles.

#### 133.5.1 Registering Services

The Mediator must track bundles that are wired to its osgi.extender=osgi.serviceloader.registrar capability. These are called the managed bundles. For all managed bundles the Mediator must enumerate all osgi.serviceloader capabilities and register selected Service Providers as OSGi services. A Service Provider is selected by an osgi.serviceloader capability when:
• The advertised Service Type matches the corresponding osgi.serviceloader capability’s Service Type, and
• The register directive is absent, or
  • The register directive contains the fully qualified name of the Service Provider.

An osgi.serviceloader capability that selects a Service Provider is said to decorate that Service Provider. A capability can decorate multiple Service Providers of the same Service Type and the same Service Provider can be decorated by different capabilities. Figure 133.2 depicts the resulting relations and their cardinalities since the relations are non-trivial.

![Cardinality Service Type Diagram]

The OSGi service for each selected Service Provider must be registered under the advertised Service Type of the Service Provider, which must match the Service Type specified in the capability.

133.5.2 OSGi Service Factory

The Mediator must register an OSGi service factory with the bundle context of the Service Provider’s bundle. The OSGi service factory must be implemented such that it creates a new instance for each bundle that gets the service. This behavior is similar, though not quite identical, to the ServiceLoader.load() method that gives each consumer a separate instance of the service. The difference is that different users inside a bundle will share the same instance.

Each service registration is controlled by a decorating osgi.serviceloader capability. The attributes on this capability must be registered with the OSGi service as service properties, except for:

• Private - Private properties, property names that start with a full stop (‘.’) must not be registered.

The following service property must be registered, overriding any identical named properties in the decorating capability:

• serviceloader.mediator - (Long) The bundle id of the mediator.

The Mediator should not verify class space consistency since the OSGi framework already enforces this as long as the publishing capability specifies the uses directive.

Any services registered in the OSGi Service Registry must be unregistered when the Service Provider’s bundle is stopped or the Mediator is stopped.

133.5.3 Service Loader and Modularity

The Service Loader API causes issues in a modular environment because it requires a class loader that has wide visibility. In a modular environment like OSGi the Consumer, the Service Type, and the Service Provider can, and should, all reside in different modules because they represent different concerns. Best practice requires that only the Service Type is shared between these actors. However,
for the Service Loader to work as it was designed the Consumer must provide a class loader that has visibility of the Service Provider. The Service Provider is an implementation class, exporting such classes is the anathema of modularity. However, since the standard JRE provides application wide visibility this was never a major concern.

The simplest solution is to make the Service Loader aware of OSGi, its API clear is mappable to the OSGi service layer. However, the Service Loader is not extensible. The result is that using the Service Loader in OSGi fails in general because the Service Loader is unable to find the Service Providers. The issues are:

- The use of the Thread Context Class Loader (TCCL) is not defined in an OSGi environment. It should be set by the caller and this cannot be enforced. The multi threaded nature of OSGi makes it hard to predict what thread a Consumer will use, making it impossible to set an appropriate TCCL outside the Consumer.
- A bundle cannot import META-INF/services since the name is not a package name. Even if it could, the OSGi framework can only bind a single exporter to an importer for a given package. The Service Loader API requires access to all these pseudo-packages via the Class Loader's getResources method, the technique used to find Service Providers.
- Instantiating a Service Provider requires access to internal implementation classes, by exporting these classes, an implementing bundle would break its encapsulation.
- If a Service Provider was exported then importing this class in a Consumer bundle would couple it to a specific implementation package; this also violates the principle of loose coupling.
- The Service Loader API does assume an eternal life cycle, there is no way to signal that a Service Provider is no longer available. This is at odds with the dynamic bundle life cycle.

133.5.4 Processing Consumers

Consumers are not written for OSGi and require help to successfully use the Service Loader API. It is the Mediator's responsibility to ensure that bundles that are wired to published Service Types have access to these Service Provider's instances through the Service Loader API.

This specification does not define how this is done. There are a number of possibilities and it is up to the Mediator to provide the guarantee to the Consumer that it has been properly processed.

A Mediator must only process Consumer's bundles that are wired to the osgi.extender capability for the osgi.serviceloader.processor extender. Since Consumers must require this extender capability with the default cardinality of 1 there can at most be one extender wired to a Consumer.

133.5.5 Visibility

The Mediator must process the Consumer bundle in such a way that when the Consumer uses the Service Loader API it receives all the Service Providers of bundles that:

- Provide one or more osgi.serviceloader capabilities for the requested Service Type, and
- Are not type space incompatible with the requestor for the given Service Type, and
- Either the Consumer has no osgi.serviceloader requirements or one of its requirements is wired to one of the osgi.serviceloader capabilities.

The Mediator must verify that the Consumer has Service Permission GET for the given Service Type since the Consumer uses the Service Type as a service. This specification therefore reuses the Service Permission for this purpose. The check must be done with the ServicePermission(String,String) constructor using the bundle's Access Control Context or the bundle's hasPermission method.

133.5.6 Life Cycle

There is a life cycle mismatch between the Service Loader API and the dynamic OSGi world. A Service Loader provides a Consumer with an object that could come from a bundle that is later stopped
and/or refreshed. Such an object becomes *stale*. Mediators should attempt to refresh bundles that have access to these stale objects.

### 133.6 osgi.serviceloader Namespace

The `osgi.serviceloader` Namespace:

- Allows the Consumer's bundle to require the presence of a Service Provider for the required Service Type.
- Provides the service properties for the service registration.
- Indicates which Service Providers should be registered as an OSGi service.

The namespace is defined in the following table and [ServiceLoaderNamespace](#), see [Common Namespaces Specification](#) on page 633 for the legend of this table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>M/O</th>
<th>Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>osgi.serviceloader</td>
<td>CA</td>
<td>M</td>
<td>String</td>
<td>qname</td>
<td>The Service Type's fully qualified name.</td>
</tr>
<tr>
<td>*</td>
<td>CA</td>
<td>O</td>
<td></td>
<td></td>
<td>Additional matching attributes are permitted. These attributes will be registered as custom service properties unless they are private (start with a full stop).</td>
</tr>
<tr>
<td>register</td>
<td>CD</td>
<td>O</td>
<td>String</td>
<td>qname</td>
<td>Use this capability to register a different Service Factory under the Service Type for each selected Service Provider.</td>
</tr>
</tbody>
</table>

An Service Provider is selected if the Service Type is the advertising Service Type and the Service Provider's fully qualified name matches the given name. If no register directive is present all advertised Service Providers must be registered. To register no Service Providers, because the capability must only be used to publish, provide an empty string.

### 133.7 Use of the osgi.extender Namespace

This section specifies the extender names for Mediators. They are used by both by Consumer and Service Provider bundles to ensure that a Mediator is present. Both names are defined for the general `osgi.extender` namespace in [osgi.extender Namespace](#) in [OSGi Core Release 6](#).

The `osgi.extender` namespace requires the use of an *extender name*, the name of the Mediator extenders is:

- `osgi.serviceloader.processor`
- `osgi.serviceloader.registrar`

The version is for this specification is in both cases: 1.0.0
133.8 Security

133.8.1 Mediator

The Mediator will require significant permissions to perform its tasks. First, it will require access to the Bundle Context of the Service Provider bundle, which means it must have Admin Permission:

AdminPermission[<Service Provider Bundles>, CONTEXT|METADATA|CLASS]

Since it will have to register on behalf of the Service Provider bundle it must have complete liberty to register services:

ServicePermission[<Service Type>, REGISTER]

Depending on the way the Consumers are processed additional requirements may be necessary. The Mediator connects two parties, it must ensure that neither party will receive additional permissions.

133.8.2 Consumers

Consumers must have:

ServicePermission[<Service Type>, GET]
PackagePermission[<Service Type's package>, IMPORT]
CapabilityPermission["osgi.extender", REQUIRE]
CapabilityPermission["osgi.serviceloader", REQUIRE]

The Mediator must ensure that the Consumer has the ServicePermission before it provides the instance. It must use the Bundle Context hasPermission method or the bundle's Access Control Context to verify this.

133.8.3 Service Providers

Service Providers must have:

ServicePermission[<Service Type>, REGISTER]
PackagePermission[<Service Type's package>, IMPORT]
CapabilityPermission["osgi.extender", REQUIRE]
CapabilityPermission["osgi.serviceloader", PROVIDE]

The Mediator must ensure that the Service Provider has the ServicePermission before it provides the instance. It must use the Bundle Context hasPermission method or the bundle's Access Control Context to verify this.

133.9 org.osgi.service.serviceloader

Service Loader Mediator 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.serviceloader; version="[1.0,2.0)"
Example import for providers implementing the API in this package:
Import-Package: org.osgi.service.serviceloader; version="[1.0,1.1)"

133.9.1 public final class ServiceLoaderNamespace
extends Namespace

Service Loader Capability and Requirement Namespace.
This class defines the names for the attributes and directives for this namespace.
All unspecified capability attributes are of one of the following types:

- String
- Version
- Long
- Double
- List<String>
- List<Version>
- List<Long>
- List<Double>

and are used as arbitrary matching attributes for the capability. The values associated with the specified directive and attribute keys are of type String, unless otherwise indicated.

All unspecified capability attributes, unless the attribute name starts with dot (\' . \'), are also used as service properties when registering a Service Provider as a service.

Concurrency Immutable

133.9.1.1 public static final String CAPABILITY_REGISTER_DIRECTIVE = "register"

The capability directive used to specify the implementation classes of the service. The value of this attribute must be of type List<String>.

If this directive is not specified, then all advertised Service Providers that match the service type name must be registered. If this directive is specified, then only Service Providers that match the service type name whose implementation class is contained in the value of this attribute must be registered. To not register a service for this capability use an empty string.

133.9.1.2 public static final String SERVICELOADER_NAMESPACE = "osgi.serviceloader"

Namespace name for service loader capabilities and requirements.
Also, the capability attribute used to specify the fully qualified name of the service type.

133.10 References

[1] Java Service Loader API
http://docs.oracle.com/javase/6/docs/api/java/util/ServiceLoader.html
134 Subsystem Service Specification

Version 1.1

134.1 Introduction

The OSGi Core Release 6 specifies a life-cycle model where bundles can be installed, resolved, and started in order to provide their own classes and services as well as use those provided by other bundles in the system. In the core framework, the bundle is the coarsest deployment unit that a management agent is able to work with; however, oftentimes it is necessary to work with collections of bundles and other types of resources, such as subsystems and implementation specific resources. For example, a collection of bundles may correspond to a particular feature of a middleware product, such as a Web container. The applications deployed to that container may also be developed as a collection of bundles that an administrator is required to manage as a consistent whole. The Subsystems specification provides a declarative model for defining resource collections, including bundles, and an API for installing and managing those collections of resources.

Many use cases only require unscoped resource collections where all provided capabilities are freely exported to and all required capabilities are freely imported from the system. However, in some cases, it is important to allow the exporting of provided capabilities to be scoped such that they can only be used by a subset of resources in the system. It may also be necessary to restrict the importing of required capabilities from outside the collection to ensure its internal capabilities are always preferred over capabilities outside the collection. For example, applications running on a Web application server or in a cloud environment may be deployed to the same server instance. The side-effects of co-locating applications on the same server must be minimized, and scoping is used to ensure each application does not use the classes and services of the others.

The framework provides hooks for influencing resolution, and access to bundles and services. These framework hooks may be used to implement scoping for a collection of bundles. The Subsystems specification provides a higher-level declarative model for defining scoping for collections of resources, including bundles.

When deploying a collection of bundles in an OSGi framework, gaps can exist between the requirements of the bundles and the capabilities provided by the target runtime. Management agents are responsible for ensuring additional bundles are installed to plug these gaps such that the collection of bundles will resolve and run. The Resolver Service Specification on page 639 and Repository Service Specification on page 531 help management agents address these needs but do not cover how deployment works for resource collections, especially when those collections are scoped. Scoping affects requirement and capability resolution and therefore affects the choice of resources. The Subsystems specification defines resolution and provisioning rules to help management agents consistently deploy collections of resources. The specification also defines a format for developers and testers to provide predetermined deployment resolutions to help ensure consistency between test and production environments.

134.1.1 Essentials

- **Collections** - Allow the management of a collection of resources as a whole.
- **Scoping** - Provide support for isolating resources in the collection such that a subset of their capabilities (for example packages and services) are available to satisfy requirements outside the Sub-
system and a subset of their requirements are able to resolve to capabilities provided outside the Subsystem.

- **Sharing** - Allow Scoped Subsystems to share their resources with others and share resources from others.
- **Dynamic** - Provide life cycle information to users of Subsystems and be able to react to changes in the state of the environment in which a Subsystem is deployed.
- **Flexible** - Enable a flexible definition with subsequent resolution to determine the resources to be used.
- **Deterministic** - Enable the choice of resources deployed for a Subsystem to be determined ahead of deployment.
- **Life-cycle** - Define a life cycle for a Subsystem describing how it affects the Subsystem's resources and allow the life cycle to be observed.
- **Reflective** - Allow discovery of runtime structural and state information.
- **Resolution** - Allow the resolution of a flexible definition during installation to determine the resources to be used.
- **Recursive** - Allow Subsystems to be defined in terms of other Subsystems.

### 134.1.2 Entities

- **Subsystem** - A collection of resources, such as bundles, or other Subsystems, administered as a whole through a Subsystem service.
- **Subsystem Manifest** - A manifest used to provide a Subsystem definition.
- **Deployment Manifest** - A manifest used to provide a deployment definition for a Subsystem. The definition identifies the exact resources to be deployed for the Subsystem.
- **Subsystem Archive** - A zip file with an .esa extension that describes a Subsystem definition. It may include the Subsystem Manifest, Deployment Manifest, or resource files that constitute the Subsystem.
- **Resource** - An element which may be used in the composition of a Subsystem, such as a bundle or another Subsystem.
- **Repository** - A service that is used to discover a Subsystem's content and dependencies. The repository service is described in the *Repository Service Specification* on page 531.
- **Resolver** - A service used to resolve requirements against capabilities to determine the resources required by a Subsystem. The resolver service is described in the *Resolver Service Specification* on page 639.
- **Constituent** - A resource that belongs to one or more Subsystems.
**134.1.3 Synopsis**

The OSGi framework does not provide any support for managing collections of resources. Management of collections of resources is enabled by a Subsystems implementation. When a Subsystems implementation is installed into the framework, it registers a Subsystem service. This service represents the framework as the **Root Subsystem**, which is a Subsystem that provides the capability to install and manage other **child Subsystems**, and is the parent of those Subsystems, but does not itself have a parent.

A Subsystem Archive provides a definition of a Subsystem that is read by the Subsystem implementation as part of installation. The Subsystem is packaged in a Subsystem Archive (.esa) file which is the Subsystem equivalent of a bundle .jar file. The Subsystem definition can be described using a Subsystem Manifest or defaulted based on the name and contents of the Subsystem Archive. Installing a new Subsystem results in another Subsystem service being registered to represent that Subsystem in the runtime. Each Subsystem service enables management and reflection of the Subsystem it represents.

A Subsystem Manifest allows flexibility in the identification of the Subsystem's content resources through version ranges and optionality. The exact versions to be deployed and any required **dependencies** (resources required to satisfy unresolved requirements of the Subsystem's content resources) can be identified in an optional Deployment Manifest. The corresponding resource binaries can be packaged in the Subsystem Archive, or found in a repository. Depending on the type of Subsystem the Subsystem Manifest may describe a **sharing policy** for the Subsystem, such as the packages or services the Subsystem exports or imports. The Deployment Manifest also describes the **sharing policy** details for the Subsystem and is defined by the type of Subsystem and the **sharing policy** described in the Subsystem Manifest.

A Subsystem that does not have a Deployment Manifest has its deployment details calculated during installation. This may be done using the **Resolver Service Specification** on page 639, if available. The starting set of requirements to be resolved are those identifying the Subsystem content (that is, requirements for content resources). The Subsystems implementation provides a resolve context that implements the policy for the Subsystem and consults the configured Repository services to find candidates to satisfy requirements.

This resolve context can also represent the target deployment environment, which might be a live framework, or a static definition of a target runtime. The resulting resolution is used to determine the exact resources to provision, equivalent to those identified in the deployment manifest. If any...
134.2 Subsystems

This specification defines a unit of installation called a Subsystem. A Subsystem is comprised of resources, including OSGi bundles and other Subsystems, which together can provide functions to end users.

A Subsystem is deployed as a Subsystem Archive (.esa) file. Subsystem Archives are used to store Subsystems and optionally their resources in a standard ZIP-based file format. This format is defined in [4] Zip File Format. Subsystems normally use the Subsystem Archive extension of .esa but are not required to. However there is a special MIME type reserved for OSGi Subsystems that can be used to distinguish Subsystems from normal ZIP files. This MIME type is:

```
application/vnd.osgi.subsystem
```

The type is defined in [5] IANA application/vnd.osgi.subsystem. A Subsystem is a ZIP file that:

- Contains zero or more resources. These resources may be OSGi bundles or other Subsystems. Subsystems may be nested or included to any depth.
- Contains an optional Subsystem Manifest named OSGI-INF/SUBSYSTEM.MF. The Subsystem Manifest describes the contents of the Subsystem Archive and provides information about the Subsystem. The Subsystem Archive uses headers to specify information that the Subsystems implementation needs to install, resolve and start the Subsystem correctly. For example, it can state the list of content resources that comprise the Subsystem and the Subsystem's type.
- Contains an optional Deployment Manifest file named OSGI-INF/DEPLOYMENT.MF. The Deployment Manifest describes the content resources, dependencies, and sharing policy that need to be provisioned to satisfy the Subsystem definition and ultimately allow it to resolve at runtime.

The Subsystem and Deployment Manifest follow the JAR manifest format (version 1.0), but with the following relaxed rules:

- No limit on the line length. Lines are allowed to exceed the JAR manifest maximum of 72 bytes.
- The last line is not required to be a carriage-return new-line combination.
- There is only one section in the manifest (the main section). A Subsystems implementation is free to ignore other sections of the manifest.
Once a Subsystem is started, its functionality is provided. Depending on the type of Subsystem it may expose capabilities, such as packages and services, to other resources installed in the OSGi framework.

### 134.2.1 Subsystem Manifest Headers

A Subsystem can carry descriptive information about itself in the Subsystem manifest file contained in its Subsystem Archive under the name `OSGI-INF/SUBSYSTEM.MF`. This specification defines Subsystem manifest headers, such as `Subsystem-SymbolicName` and `Subsystem-Version`, which Subsystem developers use to supply descriptive information about a Subsystem. A Subsystem implementation must:

- Process the main section of the manifest. Any other sections of the manifest can be ignored.
- Ignore unknown manifest headers. The Subsystem developer can define additional manifest headers as needed.
- Ignore unknown attributes and directives.

All specified manifest headers are listed in the following sections. All headers are optional. Example values are provided to help explain each header (e.g. `Export-Package: org.acme.logging; version=1.0`).

#### 134.2.1.1 Export-Package: org.acme.logging; version=1.0

The `Export-Package` header declares the exported packages for a Scoped Subsystem. See [Export-Package](#) on page 606.

#### 134.2.1.2 Import-Package: org.osgi.util.tracker; version="[1.4, 2.0)"

The `Import-Package` header declares the imported packages for a Scoped Subsystem. See [Import-Package](#) on page 606.

#### 134.2.1.3 Preferred-Provider: com.acme.logging

The `Preferred-Provider` header declares a list bundles and Subsystems which are the providers of capabilities that are preferred when wiring the requirements of a Scoped Subsystem. See [Preferred-Provider Header](#) on page 581.

#### 134.2.1.4 Provide-Capability: com.acme.dict; from=nl; to=de; version=Version=1.2

The `Provide-Capability` header declares the capabilities exported for a Scoped Subsystem. See [Resource and Wiring](#).

#### 134.2.1.5 Require-Bundle: com.acme.chess; version="[1.0, 2.0)"

The `Require-Bundle` header declares the required bundles for a Scoped Subsystem. See [Require-Bundle](#) on page 607.

#### 134.2.1.6 Require-Capability: osgi.ee; filter:="(osgi.ee=*)"

The `Require-Capability` header declares the required capabilities for a Scoped Subsystem. See [Resource and Wiring](#).

#### 134.2.1.7 Subsystem-Category: osgi, test, nursery

The `Subsystem-Category` header identifies the categories of the subsystem as a comma-delimited list.

#### 134.2.1.8 Subsystem-ContactAddress: 2400 Oswego Road, Austin, TX 74563

The `Subsystem-ContactAddress` header identifies the contact address where problems with the subsystem may be reported; for example, an email address.
134.2.1.9 **Subsystem-Content: com.acme.logging**
The Subsystem-Content header lists requirements for resources that are considered to be the contents of this Subsystem. See **Subsystem-Content Header** on page 579.

134.2.1.10 **Subsystem-Copyright: OSGi (c) 2014**
The Subsystem-Copyright header identifies the subsystem's copyright information.

134.2.1.11 **Subsystem-Description: The ACME Account Admin Application**
The Subsystem-Description header defines a human-readable description for this Subsystem, which can potentially be localized.

134.2.1.12 **Subsystem-DocURL: http://www.example.com/Firewall/doc**
The Subsystem-DocURL header identifies the subsystem's documentation URL, from which further information about the subsystem may be obtained.

134.2.1.13 **Subsystem-ExportService: org.acme.billing.Account; filter:="(user=bob)"**
The Subsystem-ExportService header specifies the exported services for a Scoped Subsystem. See **Subsystem-ExportService** on page 607.

134.2.1.14 **Subsystem-Icon: /icons/acme-logo.png; size=64**
The optional Subsystem-Icon header provides a list of URLs to icons representing this subsystem in different sizes. The following attribute is permitted:

- **size** - (integer) Specifies the size of the icon in pixels horizontal. It is recommended to always include a 64x64 icon.

The URLs are interpreted as relative to the subsystem archive. That is, if a URL with a scheme is provided, then this is taken as an absolute URL. Otherwise, the path points to an entry in the subsystem archive file.

134.2.1.15 **Subsystem-ImportService: org.acme.billing.Account; filter:="(type=premium)"**
The Subsystem-ImportService header specifies the imported services for a Scoped Subsystem. See **Subsystem-ImportService** on page 607.

134.2.1.16 **Subsystem-License: http://www.opensource.org/licenses/jabberpl.php**
The Subsystem-License header provides an optional machine readable form of license information. The purpose of this header is to automate some of the license processing required by many organizations like for example license acceptance before a subsystem is used. The header is structured to provide the use of unique license naming to merge acceptance requests, as well as links to human readable information about the included licenses. This header is purely informational for management agents and must not be processed by the Subsystems implementation.

The syntax for this header is as follows:

```
Subsystem-License ::= '<<EXTERNAL>>' |
  ( license ( ',' license ) * )
license ::= name ( ';' license-attr ) *
license-attr ::= description | link
description ::= 'description' '=' string
link ::= 'link' '=' <url>
```

This header has the following attributes:

- **name** - Provides a globally unique name for this license, preferably world wide, but it should at least be unique with respect to the other clauses. The magic name <<EXTERNAL>> is used to indi-
cate that this artifact does not contain any license information but that licensing information is
provided in some other way. This is also the default contents of this header.

Clients of this subsystem can assume that licenses with the same name refer to the same license.
This can for example be used to minimize the click through licenses. This name should be the
canonical URL of the license, it must not be localized by the translator. This URL does not have to
exist but must not be used for later versions of the license. It is recommended to use URLs from
[9] Open Source initiative. Other licenses should use the following structure, but this is not man-
dated:

http://<domain-name>/licenses/
   <license-name>-<version>.<extension>

- description · (optional) Provide the description of the license. This is a short description that is
  usable in a list box on a UI to select more information about the license.
- link · (optional) Provide a URL to a page that defines or explains the license. If this link is absent,
  the name field is used for this purpose. The URL is relative to the root of the bundle. That is, it is
  possible to refer to a file inside the bundle.

If the Subsystem-License statement is absent, then this does not mean that the subsystem is not li-
censed. Licensing could be handled outside the subsystem and the <<EXTERNAL>> form should be as-
sumed. This header is informational and may not have any legal bearing. Consult a lawyer before
using this header to automate licensing processing.

134.2.1.17 Subsystem-Localization: OSGI-INF/l10n/subsystem

The Subsystem-Localization header identifies the default base name of the localization proper-
ties files contained in the subsystem archive. The default value is OSGI-INF/l10n/subsystem.
Translations are therefore, by default, OSGI-INF/l10n/subsystem_de.properties , OSGI-INF/l10n/
subsystem_nl.properties, and so on. The location is relative to the root of the subsystem archive.
See Subsystem-Localization Header on page 574

134.2.1.18 Subsystem-ManifestVersion: 1

The Subsystem-ManifestVersion header defines that the Subsystem follows the rules of a Subsys-
tems Specification. It is 1 (the default) for this version of the specification. Future versions of the
Subsystems Specification can define higher numbers for this header.

134.2.1.19 Subsystem-Name: Account Application

The Subsystem-Name header defines a short, human-readable name for this Subsystem which may
be localized. This should be a short, human-readable name that can contain spaces.

134.2.1.20 Subsystem-SymbolicName: com.acme.subsystem.logging

The Subsystem-SymbolicName header specifies a non-localizable name for this Subsystem. The Sub-
system symbolic name together with a version identify a Subsystem Definition though a Subsystem
can be installed multiple times in a framework. The Subsystem symbolic name should be based on
the reverse domain name convention. See Subsystem-SymbolicName Header on page 572.

134.2.1.21 Subsystem-Type: osgi.subsystem.application

The Subsystem-Type header specifies the type for this Subsystem. Three types of Subsystems must
be supported: osgi.subsystem.application, osgi.subsystem.composite and osgi.subsystem.feature.
See Subsystem Identifiers and Type on page 572 for details about the three different types of Subsys-
tems. See Subsystem-Type Header on page 572 for more information about the values for the Sub-
system-Type header.

134.2.1.22 Subsystem-Vendor: OSGi Alliance

The Subsystem-Vendor header contains a human-readable description of the subsystem vendor.
134.2.1.23 Subsystem-Version: 1.0

The Subsystem-Version header specifies the version of this Subsystem. See Subsystem-Version Header on page 572.

134.2.2 Subsystem Identifiers and Type

A Subsystem is identified by a number of names that vary in their Scope:

- **Subsystem identifier** - A long that is a Subsystems implementation assigned unique identifier for the full lifetime of an installed Subsystem, even if the framework or the Subsystem's implementation is restarted. Its purpose is to distinguish Subsystems installed in a framework. Subsystem identifiers are assigned in ascending order to Subsystems when they are installed. The `getSubsystemId()` method returns a Subsystem's identifier.

- **Subsystem location** - A name assigned by a management agent to a Subsystem during the installation. This string is normally interpreted as a URL to the Subsystem Archive but this is not mandatory. Within a particular framework, a Subsystem location must be unique. A location string uniquely identifies a Subsystem. The `getLocation()` method returns a Subsystem's location.

- **Subsystem Symbolic Name and Subsystem Version** - A name and version assigned by the developer. The combination of a Subsystem symbolic name and Subsystem version is intended to provide a globally unique identifier for a Subsystem Archive or Subsystem definition. The `getSymbolicName()` method returns the assigned Subsystem name. The `getVersion()` method returns the assigned version. Though this pair is intended to be unique, it is developer assigned and there is no verification at runtime that the pair uniquely identifies a Subsystem Archive. It is possible to install a Subsystem multiple times as long as the multiple Subsystem symbolic name and version pairs are isolated from each other by Subsystem sharing policies.

134.2.3 Subsystem-SymbolicName Header

The Subsystem-SymbolicName header specifies the symbolic name of the Subsystem. The Subsystem-SymbolicName header may also specify arbitrary matching attributes. Subsystem-SymbolicName is an optional header; the default value is derived as described in Deriving the Subsystem Identity on page 573.

The Subsystem-SymbolicName header must conform to the following syntax:

```
Subsystem-SymbolicName ::= symbolic-name(';'<; parameter >)*
```

No directives are defined by this specification for the Subsystem-SymbolicName header. The header allows the use of arbitrary attributes that can be required by the Subsystem-Content header.

134.2.4 Subsystem-Version Header

The Subsystem-Version header is optional and must conform to the following syntax:

```
Subsystem-Version ::= version
```

If the Subsystem-Version header is not specified then the default value is derived as described in Deriving the Subsystem Identity on page 573.

134.2.5 Subsystem-Type Header

The Subsystem-Type header specifies the type of the Subsystem. Three types of Subsystems are defined by this specification:

- osgi.subsystem.application
- osgi.subsystemcomposite
- osgi.subsystem.feature
See Subsystem Types on page 608 for details about the three different types of Subsystems. Subsystem-Type is an optional header; the default value is osgi.subsystem.application.

The following directive must be recognized for the Subsystem-Type header:

- provision-policy - (rejectDependencies|acceptDependencies) Directive used to declare if the Subsystem is willing to accept dependencies as constituents. A constituent is the term used to refer to a resource that belongs to one or more Subsystems. It can belong to a Subsystem as a result of being listed as content or, as described here, can have been provisioned into the Subsystem as a dependency. The default policy is rejectDependencies. See Accepting Dependencies on page 585 for installing and tracking dependencies. The value acceptDependencies must not be used for Feature Subsystems. If a Feature Subsystem attempts to use the acceptDependencies policy then the Subsystem installation must fail. If the provision-policy directive is declared and has any other value besides acceptDependencies or rejectDependencies then the Subsystem installation must fail.

134.2.6 Deriving the Subsystem Identity

If the Subsystem-SymbolicName and Subsystem-Version are not specified, then the following rules are defined for deriving the values of the Subsystem’s symbolic name and version. If not otherwise specified, the default value of the version is 0.0.0.

When installing a Subsystem, the following URI syntax must be used as the location string in order to specify default values.

```
subsystem-uri ::= 'subsystem://' url? '?' params?
params ::= param ( '&' param )*
param ::= key '=' value
key ::= unreserved | escaped
value ::= unreserved | escaped
url ::= <see [6] RFC 1738 Uniform Resource Locators>
escaped ::= <see [7] Uniform Resource Identifiers (URI): Generic Syntax>
unreserved ::= <see [7] Uniform Resource Identifiers (URI): Generic Syntax>
```

The query parameters represent Subsystem Manifest header names and values. Implementations must support the Subsystem-SymbolicName and Subsystem-Version parameters. Implementations can support additional parameters but must fail the installation if any unsupported parameters are included.

As an example, the following Subsystem URI has an embedded URL of:

```
http://www.foo.com/sub#system.esa
```

It includes a default symbolic name of com.acme.foo and default version of 1.0.0.

```
subsystem://http://www.foo.com%2Fsub%52system.esa? «
Subsystem-SymbolicName=com.acme.foo&Subsystem-Version=1.0.0
```

When installing a Subsystem containing other Subsystem Archives with no symbolic name or version, values will be derived from the Subsystem Archive file or resource name. The syntax is as follows:

```
subsystem-archive ::= symbolic-name ( '@'version ) '.esa'
```

If the symbolic name is not provided in the manifest and cannot be computed by any other means then the Subsystem must fail to install.

134.2.7 Subsystem Identity Capability

The Subsystem’s symbolic name, version, and the arbitrary matching attributes specified on the Subsystem-SymbolicName header compose the osgi.identity capability for a Subsystem re-
source. The osgi.identity capability is provided by a Subsystem resource when contained within a Repository service, see Repository Service Specification on page 531. For example:

Subsystem-SymbolicName: org.acme.billing;category=banking
Subsystem-Version:      1.0.0
Subsystem-Type:         osgi.subsystem.composite

The above headers are used to declare a Subsystem of type osgi.subsystem.composite with the symbolic name of org.acme.billing, version of 1.0.0 and arbitrary matching attribute category of banking. This information will also be reflected in the osgi.identity capability of the Subsystem's Resource (org.osgi.resource.Resource). The following osgi.identity capability would be generated for a Subsystem resource from the above headers using the Provide-Capability header syntax. For example:

osgi.identity;
  osgi.identity=org.acme.billing;
  version:Version=1.0.0;
  type=osgi.subsystem.composite;
  category=banking

This allows for requirements to be used to search a repository for Subsystems. The following requirement could be used to search for all Subsystems of type osgi.subsystem.composite using the Require-Capability header syntax. For example:

osgi.identity; filter:="(type=osgi.subsystem.composite)"

### 134.2.8 Subsystem-Localization Header

For consistency and ease of comprehension, the design for localizing subsystem manifest headers follows the approach used by bundles.

#### 134.2.8.1 Localization Properties

A localization entry contains key/value entries for localized information. All headers in a subsystem's manifest can be localized. However, the subsystems implementation must always use the non-localized versions of headers that have subsystem semantics. Note that the use of localization on certain such headers, such as Subsystem-SymbolicName, may cause errors as a value with a % sign will not be valid.

A localization key can be specified as the value of a subsystem's manifest header using the following syntax:

```
header-value ::= '%'text
text ::= < any value which is both a valid manifest header value and a valid property key name >
```

For example, consider the following subsystem manifest entries:

Subsystem-Name: %acme subsystem
Subsystem-Description: %acme description
Subsystem-SymbolicName: acmeSubsystem
Acme-Defined-Header: %acme special header

User-defined headers can also be localized. Spaces in the localization keys are explicitly allowed.

The previous example manifest entries could be localized by the following entries in the manifest localization entry OSGI-INF/l10n/subsystem.properties.

```
# subsystem.properties
acme\ subsystem=The ACME Subsystem
acme\ description=The ACME Subsystem provides all of the ACME \ services
```
The above manifest entries could also have French localizations in the manifest localization entry:

```
OSGI-INF/l10n/subsystem_fr_FR.properties.
```

### 134.2.8.2 Locating Localization Entries

The Subsystems implementation must search for localization entries by appending suffixes to the localization base name according to a specified locale and finally appending the .properties suffix. If a translation is not found, the locale must be made more generic by first removing the variant, then the country and finally the language until an entry is found that contains a valid translation. For example, looking up a translation for the locale `en_GB_welsh` will search in the following order:

```
OSGI-INF/l10n/subsystem_en_GB_welsh.properties
OSGI-INF/l10n/subsystem_en_GB.properties
OSGI-INF/l10n/subsystem_en.properties
OSGI-INF/l10n/subsystem.properties
```

### 134.3 Subsystem Region

A **Region** provides isolation for a group of one or more Subsystems. Each Subsystem installed must be a member of one and only one Region. A Region consists of one and only one Scoped Subsystem and optionally a set of Unscoped Subsystems. Every Region has one and only one Parent Region, with the exception of the Root Region which has no Parent Region. The Region parent → child connections form the **Region Tree**, which by definition contains no cycles.

Each Region, except the Root Region, has a **sharing policy** associated with it which is defined by a Scoped Subsystem. A sharing policy consists of two parts:

- *Export Policy* - Defines the set of capabilities provided by the constituents contained in the Region that are made available to the parent Region.
- *Import Policy* - Defines the set of capabilities which are available in the parent Region that are made available to the child Region.

Figure 134.2 illustrates a set of Regions that contain capabilities and requirements for a capability. For the purposes of this illustration the Subsystems and resources have been omitted.

**Figure 134.2 Regions and Import/Export**

In this example some constituent of Region $S_1$ provide a capability $S_1 \rightarrow X$. The $S_1$'s sharing policy exports the capability $S_1 \rightarrow X$ to its parent $RootRegion$. The $RootRegion$ contains a constituent which has a requirement $Root \rightarrow X$. The export sharing policy of $S_1$ allows visibility to the capability $S_1 \rightarrow X$ from the $RootRegion$ which allows requirement $Root \rightarrow X$ to be satisfied by the capability $S_1 \rightarrow X$. The $S_2$ also contains a constituent which has a requirement on $S_2 \rightarrow X$. 


The sharing policy of $S_2$Region imports the capability $X$ from its parent Region $Root$Region. Since $Root$Region has visibility to the capability $S_1 \rightarrow X$ this allows $S_2$Region to also have visibility to capability $S_1 \rightarrow X$ through its import sharing policy which allows requirement $S_2 \rightarrow X$ to be satisfied by the capability $S_1 \rightarrow X$.

Sharing policies of the Regions allow for a capability to be shared across an arbitrary number of Regions. For those familiar with the Region digraph, see [8] Equinox Region Digraph, the connections between Subsystem Regions is more restrictive than what the full Region digraph specification allows. A visibility path is the path over the sharing policies of the Region tree from a requirement to a capability that allows a requirement to get wired to a capability. Since all (non-Root) Regions have one and only one Parent Region the visibility paths over the sharing policies between a requirement and a capability is limited to 0 or 1. Figure 134.3 is another figure that illustrates a capability being shared across many different Regions.

**Figure 134.3** Regions and Sharing Capabilities

In this example the capability $S_3 \rightarrow X$ is exported by the $S_3$Region sharing policy to its parent $S_1$Region. $S_1$Region also exports $X$ to its parent $Root$Region. Then $S_2$Region imports $X$ from its parent $Root$Region and finally $S_4$Region imports $X$ from its parent $S_2$Region. The visibility path from requirement $S_4 \rightarrow X$ to capability $S_3 \rightarrow X$ is the following: $S_4 \rightarrow S_2 \rightarrow Root \rightarrow S_1 \rightarrow S_3$.

Notice that in this example the $S_5$Region also has a sharing policy that imports $X$ from its parent $S_3$Region. Child Regions are allowed to import any capability to which the Parent Region has visibility. This is true even if the Parent Region does not export the capability. Regions can selectively choose what capabilities they want to expose (or export) to their Parent Region. Child Regions also can selectively choose what capabilities they want to be exposed to (or import) from their Parent Region. A Parent Region has no control over what capabilities its children have visibility. Similarly a Parent Region has no control over what capabilities a Child Region is allowed to export to the Parent Region. In other words, a Parent Region must give a Child Region everything the Child Region asks for (if the Parent Region has access to it) and a Parent Region must accept everything a Child Region offers to the Parent Region.

### 134.4 Subsystem Relationships

Subsystems installed into a framework become part of the Subsystem graph. The Subsystem graph may be thought of as is directed acyclic graph with one and only one source vertex, which is the Root Subsystem. The edges have the child as the head and parent as the tail (parent→child). This is depicted in Figure 134.4.
A Subsystem installed into or included within one or more Subsystems is called a **child Subsystem**. A Subsystem which has one or more child Subsystem(s) installed or included in it is called a **parent Subsystem**. Note that a Subsystem may be both a parent and child Subsystem. The Subsystem graph has the following rules:

- There is one and only one source vertex (i.e. a Subsystem with no parents), which is the **Root Subsystem**.
- The Root Subsystem is considered a Scoped Subsystem of type application with a provision-policy of acceptDependencies.
- The Root Subsystem has a symbolic name of org.osgi.service.subsystem.root, version 1.0.0, Subsystem identifier of 0, and a location of
  
  subsystem://?Subsystem-SymbolicName=org.osgi.service.subsystem.root& «
  Subsystem-Version=1.0.0

- All parents of a Subsystem belong to the same Region.
- An Unscoped Subsystem must belong to the same Region to which its parents belong.
- A Scoped Subsystem (other than the Root Subsystem) must belong to a child Region of the Region to which the Subsystem's parents belong.

When a Subsystem is installed using a Subsystem service `install(String)` or `install(String,InputStream)` method the **Subsystem resource** becomes a constituent of the Subsystem which the install method was called on. The **Subsystem resource** is the Subsystem Archive and may be retrieved by calling the Subsystem service `getConstituents()` method. Figure 134.4 illustrates the Root Subsystem with initial bundles A, B, SI (Subsystems implementation, may be multiple bundles), and the system bundle (identifier 0).

In Figure 134.5 Bundles A, B and SI are considered constituents of the Root Subsystem. The system bundle is also considered to be a constituent of the Root Subsystem (bundle zero). A Subsystem ser-
vice R is registered that represents the Root Subsystem. When Subsystems are installed using the Root Subsystem then these Subsystem resources become constituents of the Root Subsystem and the Subsystems become child Subsystems of the Root Subsystem. For example, Figure 134.6 illustrates the Root Subsystem with Scoped Subsystem S1 with constituent bundles C and D and Scoped Subsystem S2 with constituent bundles E and F:

**Figure 134.6** Subsystems

![Subsystems Diagram]

The two Subsystems S1 and S2 have the same parent and Subsystems S1 and S2 are children of the Root Subsystem. Figure 134.7 shows a more complicated tree that has both Scoped and Unscoped Subsystems installed. This figure omits the constituent resources and Subsystem services:

**Figure 134.7** Parent Child Relationship with Unscoped Subsystems

![Unscoped Subsystems Diagram]

### 134.4.1 Prevent Cycles and Recursion

It is possible to end up with cycles in the parent → child relationships between Subsystems contained in the same Region. Figure 134.8 illustrates this.

**Figure 134.8** Subsystems and cycles

![Subsystems and Cycles Diagram]

In this example Subsystem S1 has been installed. The Scoped Subsystem S1 has included in its constituents the Unscoped Subsystems U1 and U2. Furthermore U1 has included the Unscoped Subsystem U2 as a constituent and U2 has included the Unscoped Subsystem U1 as a constituent. This causes Unscoped Subsystem U1 to have parents S1 and U2 and Unscoped Subsystem U2 to have parents S1 and U1. There is now a cycle between the Subsystems U1 and U2. Subsystems implementations must detect this cycle and fail the installation of such a degenerative Subsystem. The top level Subsystem being installed must fail the install operation by throwing a Subsystem Exception. In this case the install operation of the S1 Subsystem must fail with a Subsystem Exception being thrown.
Cycles may also exist in the definition of Scoped Subsystems which includes other child Subsystems. Figure 134.9 illustrates this.

Figure 134.9  Scoped Subsystems and cycles

In this example the Scoped Subsystem S1 includes as a child the Scoped Subsystem S2. The S2 subsystem also includes as a child the Scoped Subsystem S1. Subsystems implementations must detect this and fail the installation of such a degenerative Subsystem. The top level Subsystem being installed must fail the install operation by throwing a Subsystem Exception. In this case the install operation of the first S1 Subsystem must fail by throwing a Subsystem Exception.

### 134.5 Determining Content

A Subsystem definition may declare different types of content resources. A Subsystems implementation may support additional types, but the following types must be supported:

- `osgi.bundle`
- `osgi.fragment`
- `osgi.subsystem.application`
- `osgi.subsystem.feature`
- `osgi.subsystem.composite`

A Subsystems implementation is free to support additional content types as value-add, but an implementation is required to fail the installation of a Subsystem which declares content resource types which are not recognized by the implementation.

The individual content resources may be specified in the following ways:

- The Subsystem-Content header, or
- The entries of the Subsystem Local Repository, see Resource Repositories on page 582.

#### 134.5.1 Subsystem-Content Header

The Subsystem-Content header contains a list of symbolic names, with optional attribute and directive assertions. Each element specifies a single resource that is to be a content resource of the Subsystem. See also Discovering Content Resources on page 583. The Subsystem-Content header must conform to the following syntax:

```
Subsystem-Content ::= resource ( ','resource )*  
resource ::= symbolic-name ( ';' parameter )*  
```

The Subsystem-Content header may specify the following directives:

- `resolution` (mandatory|optional) A mandatory content resource prevents the Subsystem from successfully installing when the constituent cannot be found (or satisfied); an optional content resource allows a Subsystem to successfully install even if the content cannot be found (or satisfied). The default value is mandatory.
- `start-order` (Integer >= 1) Specifies the start order of the content resource in relation to other content resources of the Subsystem. See Start Order on page 593.

The Subsystem-Content header may specify the following architected matching attributes as well as any arbitrary matching attributes:
• version - (Version) A version range used to select the version of the resource to use. This follows the OSGi version range scheme, including the default value of 0.0.0.
• type - Indicates the type of the content. It is recommended that a reverse domain name convention is used unless those types and their processing is standardized by the OSGi Alliance, for example bundles. The default type is osgi.bundle. A Subsystems implementation may support additional types, but the following types must be supported:
  • osgi.bundle
  • osgi.fragment
  • osgi.subsystem.application
  • osgi.subsystem.composite
  • osgi.subsystem.feature

For example, the following header specifies three Subsystem constituents:

Subsystem-Content:
  org.acme.billing.impl;
    type=osgi.bundle;
    version=1.0,
  org.acme.billing.frag;
    type=osgi.fragment;
    version=1.0,
  org.acme.billing.credit.subsystem;
    type=osgi.subsystem.composite;
    version=1.0

The above header specifies three content resources of a Subsystem:

• A bundle resource with the symbolic name org.acme.billing.impl at version 1.0 or greater
• A fragment resource with the symbolic name org.acme.billing.frag at version 1.0 or greater
• A child composite Subsystem resource with the symbolic name org.acme.billing.credit.subsystem at version 1.0 or greater

134.5.2 Subsystem-Content Requirements

Each element of the Subsystem-Content header is used to locate a resource that is to be used as content of the Subsystem. One way of describing the elements of the Subsystem-Content header is in terms of a Requirement using the osgi.identity namespace. The Requirement is defined in [3] Resource and Wiring. To illustrate, a single Subsystem-Content element:

org.acme.billing.impl;
  type=osgi.bundle;
  version=1.0

This Subsystem-Content header can be converted into the following osgi.identity Requirement with the Require-Capability syntax for illustration:

osgi.identity;
  filter:="(&
    (osgi.identity=org.acme.billing.impl)
    (type=osgi.bundle)
    (version>=1.0)
  )"

All directives specified on the Subsystem-Content header, except start-order, should be included in the Requirement. All attributes should be included in the filter directive of the Requirement. Notice that the version attribute is a range and must be converted into a proper filter. The
VersionRange.toFilter method can be used to do this conversion. All other matching attributes are treated as type String and use an equality operation in the filter. Here is an example of a more complex transformation to Requirement:

```
org.acme.billing.credit.subsystem;
  type=osgi.subsystem.composite;
  version="[1.0, 1.1]";
  category=banking;
  resolution:=optional;
  start-order:=1
```

The above Subsystem-Content element converts into the following osgi.identity Requirement:

```
osgi.identity;
  filter:="(&
    (osgi.identity=org.acme.billing.impl)
    (type=osgi.subsystem.composite)
    (&(version>=1.0)(!(version>=1.1))
    (category=banking)
  )"
  resolution:=optional
```

134.5.3 Preferred-Provider Header

The Preferred-Provider header contains a list of symbolic names, with optional attributes assertions. Each element specifies a single bundle or Subsystem resource that is to be preferred when resolving the requirements of the Subsystem content resources. The Preferred-Provider header must conform to the following syntax:

```
Preferred-Provider ::= resource (',' resource )*
resource ::= symbolic-name ( ';' attribute )*
```

The Preferred-Provider header may specify the following architected matching attributes:

- **version** (Version) A version range used to select the version of the bundle or Subsystem to use. This follows the OSGi version range scheme, including the default value of 0.0.0.
- **type** (String) Indicates the type of the provider. Valid types are:
  - osgi.bundle
  - osgi.subsystem.composite
  - osgi.subsystem.feature

The default type is osgi.subsystem.composite. Specifying an unsupported type results in an installation failure.

Each element of the Preferred-Provider header is used to locate a resource that is to be used as a preferred provider of the Subsystem. The Preferred-Provider header elements are converted to Requirements using the osgi.identity namespace just like the Subsystem-Content header except the default type is osgi.subsystem.composite. See Subsystem-Content Requirements on page 580.

Because this header influences resolution, it is only valid for it to be used on a Scoped Subsystem. If a Subsystems implementation encounters this header on an Unscoped Subsystem, it must fail the installation of the Subsystem.

The Preferred-Provider header has the effect of influencing the import policy into the Region representing the Scoped Subsystem that specified the header. If there are multiple candidate capabilities for a requirement and one or more of those capabilities is from a bundle or Subsystem identified in the Preferred-Provider header, then the Region import policy must prefer the capabilities from the preferred bundle or Subsystem.
A resource may be considered as a preferred provider only if it is a constituent of either the Scoped Subsystem's or any ancestor's Region.

### 134.5.4 Resource Repositories

When a Subsystem is installed the Subsystems implementation is responsible for provisioning resources that are associated with the Subsystem. For example, the Subsystem's content resources as well as any resources that are needed to satisfy dependencies of a Subsystem's content resources. During the Subsystem install process the Subsystems implementation uses a defined set of repositories to find the required resources needed to install a Subsystem. This set of repositories includes the following:

- **Local Repository** - Contains the resources included in the Subsystem Archive, see [Local Repository](#) on page 582.
- **System Repository** - Contains the resources currently installed, see [System Repository](#) on page 582.
- **Repository Services** - The set of repositories registered as OSGi services, see [Repository Services](#) on page 582.
- **Content Repository** - The set of resources that comprise the Subsystem content, see [Content Repository](#) on page 582.
- **Preferred Repository** - The set of resources that are considered preferred providers, see [Preferred Repository](#) on page 583.

#### 134.5.4.1 Local Repository

The Root of the Subsystem Archive contains 0 or more resources. The Subsystems implementation must read all entries (except directory entries) in the Root of the Subsystem Archive and treat each entry as a potential resource. One way of describing the resource entries contained in the Root of the Subsystem Archive is in the terms of an Repository implementation. For the purpose of this specification these resources are referred to as the Subsystem's **Local Repository**. The Local Repositories must not be registered as an OSGi Repository service. Also, it is not required that the Subsystem implementation actually implement a Local Repository as a concrete implementation of the Repository service interface.

#### 134.5.4.2 System Repository

The term **System Repository** is used to describe the set of resources that are constituents of one or more of the currently installed Subsystems. The System Repository must not be registered as an OSGi service. Also it is not required that System Repository be implemented as a concrete implementation of the Repository service. There is a single System Repository representing the resources installed in the OSGi framework.

#### 134.5.4.3 Repository Services

The repositories which are registered as Repository services, see [Repository Service Specification](#) on page 531. These Repositories are used to discover Subsystem content resources and dependencies. A Subsystems implementation searches registered Repository services by service ranking order.

#### 134.5.4.4 Content Repository

The set of content resources for a Subsystem is referred to as the Subsystem's Content Repository. Similar to the Local and System Repositories, the Content Repositories must not be registered as an OSGi service and it is not required that the Subsystems implementation actually implement a Content Repository as a concrete implementation of the Repository service. There are two types of resources that can exist in a Subsystem's Content Repository:

- **Installable Content** - A content resource which must be installed and result in a distinct resource at runtime. That is, a distinct bundle or Subsystem installation.
• **Shared Content** - A content resource which is already installed and is a constituent of one or more already installed Subsystems that belong to the same Region as the Subsystem that the Subsystem content repository is for. This resource must be reused, the Subsystems implementation must not install another instance of the resource.

Details on how the content resources are discovered for the Content Repository are discussed in *Discovering Content Resources* on page 583.

### 134.5.4.5 Preferred Repository

The set of resources which are considered preferred providers of capabilities required by a Subsystem is referred to as the Preferred Provider Repository for the Subsystem. The Preferred Provider Repository for a Subsystem must not be registered as an OSGi service and it is not required that the Subsystems implementation actually implement the Preferred Provider Repository as a concrete implementation of the Repository service.

The following steps must be followed to discover the resources of the preferred provider repository for a Subsystem:

1. The Preferred-Provider header is parsed into a list of elements where each element specifies a single `osgi.identity` requirement, see *Preferred-Provider Header* on page 581.
2. For each `osgi.identity` requirement a Requirement object is created and used to search Repositories for preferred provider resources.
3. The System Repository is searched. For each capability found in the System Repository; if the resource providing the `osgi.identity` capability is a constituent contained in the parent Region of the Scoped Subsystem's Region then the providing Resource of the Capability is considered a preferred provider and the search stops; otherwise continue to the next step.
4. The Subsystem's Local Repository is searched. If a capability is found then the providing resource is used as a preferred provider and the search stops; otherwise continue to the next step.
5. The registered Repository services are searched. If a repository service finds a capability then the providing resource is used as a preferred provider and the search stops; otherwise the preferred provider is not found.

### 134.5.5 Discovering Content Resources

When a Subsystem is installed the Subsystems implementation must determine the set of resources that compose the content of the Subsystem. The content resources of a Subsystem may be specified in the following ways:

- The values of the Subsystem-Content header must be used if it is present. See *Subsystem-Content Header* on page 579.
- The content of the Subsystem's Local Repository, if the Subsystem-Content header is not present.

When a Deployment Manifest is not present, *Pre-Calculated Deployment* on page 603, the Subsystems implementation must use this information to discover the content resources for a Subsystem as described in the following sections.

### 134.5.5.1 Declared Subsystem-Content

If the Subsystem-Content manifest header is declared then the following steps must be followed to discover the Subsystem's contents:

- The Subsystem-Content header is parsed into a list of elements where each element specifies a single `osgi.identity` requirement. For each `osgi.identity` requirement element a Requirement is created and used to search Repositories for content resources.
- If the Subsystem is a Scoped Subsystem then continue to the next step; otherwise if the Subsystem is an Unscoped Subsystem then the System Repository must be searched in order to
discover any currently installed resources that match the content Requirement. For each matching capability found it must be determined if the capability provider Resource is a constituent of a Subsystem which is in the same Region as the installing Subsystem; if so then the provider Resource must be used as an \textit{shared content} resource. If no \textit{shared content} resource is found then continue to the next step; otherwise the search stops.

- The Subsystem's Local Repository is searched to find a matching Capability for the content Requirement. If a Capability is found then the providing Resource of the Capability is used as an \textit{installable content} resource of the Subsystem. If no \textit{installable content} resource is found then continue to the next step, otherwise the search stops.
- The registered Repository services are searched to find a matching capability for the content Requirement. If a Repository finds a provider for the content requirement then the provider Resource of the capability is used as an \textit{installable content} resource of the Subsystem. If no matching provider is found then the discovery of the content resource has failed.

\section*{134.5.5.2 Use Subsystem Local Repository}

If the Subsystem-Content header is not declared then the list of content resources is defined as all the Resources within the Subsystem's Local Repository which provide an \texttt{osgi.identity} capability with the type attribute of:

- \texttt{osgi.bundle}
- \texttt{osgi.fragment}
- \texttt{osgi.subsystem.application}
- \texttt{osgi.subsystem.composite}
- \texttt{osgi.subsystem.feature}
- Any other type that is supported by the implementation.

If a resource is found to be an unsupported type then installation of the Subsystem must fail.

For Scoped Subsystems this list is used as is and each Resource is considered an installable content resource. For Unscoped Subsystems the System Repository must be searched in order to determine if there are any already installed contents resources which may be used as a \textit{shared content} resources. If no shared content resource can be found then the resource is considered an \textit{installable content} resource.

\section*{134.6 Determining Dependencies}

When a Subsystem is installed the Subsystems implementation determines the set of resources that compose the content of the Subsystem. Content resources may have requirements on capabilities that are not provided by any of the content resources for the Subsystem. When a Subsystem is installed the Subsystems implementation must determine the set of additional resources that are required in order to allow the Subsystem's content resources to resolve. These additional resources are called \textit{dependencies}. When a Subsystem is installed the set of dependencies contains two types of resources:

- \textit{Installable Dependency} - A resource which must be installed and result in a distinct resource at runtime. That is, a distinct bundle or Subsystem installation.
- \textit{Preinstalled Dependency} - A resource which is already installed and is a constituent of one or more already installed Subsystems and the required capabilities provided by the resource are accessible according to the sharing policies of the Subsystems. This resource must be reused, the Subsystems implementation must not install another instance of the resource.

When a Deployment Manifest is not present, see \textit{Pre-Calculated Deployment} on page 603, the Subsystems implementation must determine the set of dependencies for the Subsystem. To determine
the set of dependencies the Subsystems implementation should attempt to resolve the Subsystem content resources before installing the content resources. One possible way of resolving the content resources, before installing them, is to use a Resolver service, see Resolver Service Specification on page 639. This specification illustrates the behavior of dependency resource discovery using terms defined by the Resolver service. A Subsystems implementation is not required to use the Resolver service to accomplish dependency resource discovery. Other mechanisms may be used to accomplish the same goal as long as the resolution results in a valid class space. Subsystems implementations need not guarantee to find a solution to every resolution problem, but if a valid solution is not found, then discovery of the dependencies must fail, resulting in an installation failure.

In order to use the Resolver service the Subsystems implementation has to provide a ResolveContext object that represents the currently installed Subsystems and their constituent resources. This resolve context must search Repositories in the following order when searching for capabilities to satisfy content dependencies within the ResolveContext.findProviders method. The Content Repository, Preferred Provider Repository, System Repository and Local Repository must all be searched and all the results presented to the Resolver with a corresponding preference. If a result was found in these repositories, searching the registered Repository services is optional, but if no result was found yet, the registered Repository services must be searched. The order of matching capabilities presented to the Resolver must coincide with the Repository search order.

1. The Content Repository.
2. The Preferred Provider Repository.
3. The System Repository. For each matching capability found in the System Repository the Subsystems implementation must determine if the capability is accessible to the content resources of the installing Subsystem according to the sharing policy of the Subsystem. See Sharing Capabilities on page 587 for more details on Subsystem types and their sharing policies.
4. The Local Repository. This allows a Subsystem Archive to optionally include dependencies.
5. The registered Repository services.

Any matching capabilities found in the steps after step 1 above are considered to be provided by potential dependencies for the Subsystem. The capabilities found in the System Repository are provided by already installed resources, referred to as potential pre-installed dependencies. The capabilities found which are provided by other potential dependencies must be installed in order to resolve the Subsystem content resources, referred to as installable dependencies.

The Resolver's job is to select one of the potential capabilities returned by the findProviders method as the capability to satisfy a Requirement. At the end of a resolve operation a result (Map<Resource, List<Wire>>) is returned which contains the Resources that got resolved and a list containing the Wires for the resolved Resource. The Subsystems implementation uses this resolution result in order to determine which resolved Resources are content resources, pre-installed dependencies, or installable dependencies. The installable dependencies must be installed as described in Accepting Dependencies on page 585. The pre-installed dependencies must have their reference count incremented as described in Reference Count on page 592.

134.7 Accepting Dependencies

When a Subsystem is installed the Subsystems implementation must determine the set of installable dependencies as described in Determining Dependencies on page 584. The Subsystems implementation must also determine what Subsystem is willing to accept the installable dependency as a constituent, referred to as an accepted dependency constituent. A Subsystem declares that it is willing to accept dependencies as constituents by specifying the provision-policy directive with the acceptDependencies value on the Subsystem-Type header, see Subsystem-Type Header on page 572.

The acceptDependencies provision policy is useful for creating isolation layers that do not pollute parent Regions with dependencies. For example, an application container may be implemented as
a Subsystem. Such a container Subsystem could be installed into something called a kernel Subsystem. Applications are installed as Subsystems into the container Subsystem. In this case the container Subsystem would likely use the acceptDependencies provision policy so that any applications installed into the container Subsystem do not end up polluting the kernel Subsystem with the application's dependencies.

A dependency becomes an accepted dependency constituent of the Subsystem with a provision policy of acceptDependencies and that lies on the longest path between the Root Subsystem and the Subsystem being installed, inclusively. Note that a Subsystem that has acceptDependencies provision policy will accept its own installable dependencies as constituents since it lies on the longest path between the Root Subsystem and itself, inclusively.

The following figure illustrates a simple example of accepting dependency constituents. A Scoped Subsystem S2 is being installed into another Scoped Subsystem S1 and S1 has a provision-policy of acceptDependencies. When S2 is being installed the Subsystems implementation discovers content resources A and B and installable dependencies C and D. This is depicted in Figure 134.10.

**Figure 134.10 Provision Policy**

![Figure 134.10 Provision Policy](image)

In the previous example the Subsystem with a provision-policy of acceptDependencies and that lies on the longest path between the Root Subsystem and the Subsystem being installed is S1. Therefore the installable dependencies C and D become accepted dependency constituents of S1.

Figure 134.11 illustrates the same example but with S2 also having a provision-policy of acceptDependencies.

**Figure 134.11 Provision Policy**

![Figure 134.11 Provision Policy](image)

In this example the Subsystem with a provision-policy of acceptDependencies and that lies on the longest path between the Root Subsystem and the Subsystem being installed is S2 itself. Therefore the installable dependencies C and D become accepted dependency constituents of S2.

Figure 134.12 illustrates the same example but with S1 and S2 not defining the provision-policy (default is rejectDependencies). The Root Subsystem always has a provision-policy of acceptDependencies:

**Figure 134.12 Subsystems and acceptDependencies**

![Figure 134.12 Subsystems and acceptDependencies](image)
In this example the Subsystem with a provision policy of acceptDependencies and that lies on the longest path between the Root Subsystem and the Subsystem being installed is the Root Subsystem. Therefore the installable dependencies C and D become accepted dependency constituents of the Root Subsystem.

134.8 Sharing Capabilities

Scoped Subsystems define a sharing policy for the Region they are contained in. A sharing policy controls access to capabilities between parent → child Region boundaries. For Subsystems, a sharing policy is composed of two parts: an import policy and an export policy:

- **Export Policy** - Defines the set of capabilities provided by the constituents contained in the Region that are made available to the parent Region.
- **Import Policy** - Defines the set of capabilities which are available in the parent Region that are made available to the child Region.

The import sharing policies of a Subsystem's ancestor parent chain may prevent a Subsystem from accessing the required capabilities provided by a dependency. Figure 134.13 illustrates this.

**Figure 134.13 Sharing Capabilities**

In this example the C and D provide capabilities that are required by A and B respectively. The import sharing policy of S2 allows the capabilities to be imported into S2, but the import sharing policy for S1 denies the import of one of the capabilities and allows the other. In order for A or B to access capabilities provided by C or D they must traverse both the import sharing policy of S2 and S1. Ultimately S1 sharing policy prevents the necessary access to the dependencies necessary to resolve S2. Such a situation must be detected during the discovery of the installable dependencies and result in a failure to install the dependencies. This must result in the failure to install the Subsystem that requires the dependency, in this case S2.

134.8.1 Preferred Provider

The sharing policy for a Scoped Subsystem may specify a set of preferred providers. If a capability is provided by a preferred provider then that capability must be used to resolve a Scoped Subsystem's import policy. Figure 134.13 illustrates this.

**Figure 134.14 Preferred Provider**
In this example constituent B of the Root Region provides the capability X (called Root→X). Also constituent C of S2Region provides the capability S2→X. The export sharing policy of S2 policy exports the capability S2→X to its parent Region, the RootRegion. The S1Region contains a constituent A that has a requirement on S1→X. The two capabilities, Root→X and S2→X, are available to satisfy the requirement S1→X. The S1Region’s import sharing policy imports capability X and has a preferred provider of S2. This means that the capability S2→X must be used to satisfy the requirement S1→X.

134.8.2 System Capabilities

The osgi.ee and osgi.native namespaces are used by the System Bundle to describe capabilities for the Java execution environment and the native environment. These capabilities must only be provided by the System Bundle. A Subsystems implementation must allow access to the osgi.ee and osgi.native capabilities provided by the System Bundle to every Subsystem installed. This includes scoped sub systems which may or may not already have an import sharing policy configured to import these namespaces from the System Bundle.

134.9 Region Context Bundle

The Region context bundle provides a perspective from a constituent of a Subsystem contained in the Region. When a Scoped Subsystem is installed the Subsystems implementation must generate the Region context bundle and install it as a constituent bundle of the Subsystem. The Region context bundle has the following characteristics:

- Has a symbolic name of org.osgi.service.subsystem.region.context.<subsystem id>
- Version 1.0.0
- Has a location string of <subsystem location>/<subsystem id>
- Must always be allowed to resolve and start (i.e. has no requirements, imports or bundle activator).
- Has a start-level of 1 and is persistently started.
- Is installed and started before the Subsystem service is registered.

This bundle is installed and must remain active as long as the Subsystem is installed. If the Region context bundle is stopped, updated or uninstalled then the Subsystem runtime should log an error and may ensure the context bundle is installed and restarted.

When the Subsystems implementation is active it must establish the Root Subsystem, see Subsystem Service on page 595. In establishing the Root Subsystem the Subsystems implementation must ensure that there is a Region context bundle available for the Root Region. This Root Region context bundle will have a symbolic name of org.osgi.service.subsystem.region.context.0.

Typically the Region’s context bundle would be used to obtain a bundle context with the getBundleContext() method, which has a perspective as a constituent of the Region. This is useful in the following ways:

- Implementing Subsystem aware extenders. Such extenders need to be able to register listeners and monitor the inside of a Region in order to react to the constituent bundles of a Region.
- Monitoring of internal events.

134.10 Explicit and Implicit Resources

Depending on how a resource is installed the Subsystems implementation considers the resource to be either an explicit resource or implicit resource. An explicit resource is a resource whose installation was initiated by an agent outside of the Subsystems implementation. An implicit resource is a re-
source whose installation was initiated by the Subsystems implementation during the explicit installation of a Subsystem, including the content and dependencies of the explicitly installed Subsystem.

### 134.10.1 Explicit Resources

An explicit resource is a resource that was installed programmatically, by an agent outside of the Subsystems implementation, using some resource specific API. This specification defines two types of resources that can be installed explicitly:

- **Subsystem resource** - A Subsystem resource may be installed explicitly by using the Subsystem service. Note that content and dependencies of an explicitly installed Subsystem are not considered to be explicit resources themselves since they were implicitly installed by the Subsystems implementation.
- **Bundle resource** - A bundle resource may be installed explicitly by using a bundle context. This includes fragments.

### 134.10.1.1 Explicit Bundle Resources

When a bundle is installed explicitly with a bundle context, the Subsystems implementation must determine the Subsystem of which a bundle becomes a constituent. The following rules are followed when a bundle is installed explicitly with a bundle context:

1. Determine the bundle performing the install. This is the bundle whose Bundle Context is performing the install operation.
2. Determine the target Region. This is the Region to which the bundle performing the install operation belongs.
3. If a bundle with the same symbolic name and version already exists in the target Region then the bundle installation must fail unless the same location string is used. If the same location is used then the existing bundle is returned. This may be accomplished by the use of a bundle collision hook.
4. Determine the Subsystem(s) of which the bundle performing the install is a constituent. The bundle performing the install may be a shared resource. In that case the bundle performing the install is a constituent of two or more Subsystems.
5. The newly installed bundle must become a constituent of all the Subsystems of which the bundle performing the install is a constituent.

### 134.10.1.2 Explicit Subsystem Resources

When a Subsystem is installed explicitly with a Subsystem service, the Subsystems implementation must determine what Subsystem(s) the Subsystem resource and its children will become a constituent of. The following rules are followed when a Subsystem is installed:

1. Determine the target Subsystem. This is the Subsystem service which is performing the install operation or the Subsystem which includes another Subsystem as part of its content, see Determining Content on page 579.
2. Determine the target Region. This is the Region to which the target Subsystem belongs.
3. If no Subsystem resource with the same location exists then continue to the next step; otherwise do the following:
   - If the existing Subsystem is not a part of the target Region then fail the install operation by throwing a Subsystem Exception; otherwise continue to the next step.
   - If the existing Subsystem symbolic name, version and type is not the same as the Subsystem being installed then fail the install operation by throwing a Subsystem Exception; otherwise continue to the next step.
   - If the existing Subsystem is already a constituent of the target Subsystem then return the existing Subsystem from the install method; otherwise the existing Subsystem resource be-
comes a shared resource by increasing the reference count of the existing Subsystem by one, see Reference Count on page 592, and the existing Subsystem becomes a constituent of the target Subsystem; finally, the existing Subsystem is returned from the install method.

4. If no Subsystem resource with the same symbolic-name and version already exists in the target Region then the Subsystem resource being installed becomes a constituent of the Subsystem target; otherwise do the following:
   • If the existing Subsystem type is not the same as the type of the Subsystem being installed then fail the install operation by throwing a Subsystem Exception; otherwise continue to the next sub-step.
   • If the existing Subsystem is already a constituent of the target Subsystem then return the existing Subsystem from the install method; otherwise the existing Subsystem resource becomes a shared resource by increasing the reference count of the existing Subsystem by one and the existing Subsystem becomes a constituent of the target Subsystem; finally, the existing Subsystem is returned from the install method.

134.10.2 **Explicit Resource Example**

A scenario is used to illustrate the rules for determining which Subsystem an explicitly installed resource is a constituent. Figure 134.15 illustrates the Root Subsystem with initial content bundles A, S1 (Subsystems implementation) and the system bundle (id = 0) installed.

*Figure 134.15 Explicit Resource Example*

If bundle A uses its own Bundle Context to explicitly install bundle B then bundle B becomes a constituent of the Root Subsystem. If bundle A uses Subsystem R to install Scoped Subsystem S1 then the S1 resource becomes a constituent of the Root Subsystem and S1 Subsystem becomes a child of the Root Subsystem. S1 contains constituent bundles C and D. Also, if bundle B uses Subsystem R to install Scoped Subsystem S2 then the S2 resource becomes a constituent of the Root Subsystem and the S2 Subsystem becomes a child of the Root Subsystem. S2 contains constituent bundles E and F. Figure 134.16 illustrates this.

*Figure 134.16 Subsystems and Resources*

Then if bundle C uses its own Bundle Context to install bundle F (using a different location string from constituent bundle S2 → F) then the bundle becomes a constituent of S1. If bundle E uses Sub-
system service $S_2$ to install Unscoped Subsystem $U_1$ (with constituents $G$ and $H$) and installs Unscoped Subsystem $U_2$ (with constituents $H$ and $J$) then both Subsystem bundles $U_1$ and $U_2$ become constituents of $S_2$. The $S_2$ Subsystem also becomes the parent Subsystem for both $U_1$ and $U_2$ Subsystems, see Figure 134.17.

In this scenario bundle $H$ is a shared constituent of both $U_1$ and $U_2$ Subsystems. If bundle $H$ installs a bundle $K$ with its bundle context then bundle $K$ becomes a shared constituent of both $U_1$ and $U_2$ Subsystems. Also, if Subsystem service $U_1$ is used to install Scoped Subsystem $S_3$ and Subsystem service $U_2$ is also used to install Subsystem $S_3$ then $S_3$ resource becomes a shared constituent of both Unscoped Subsystems $U_1$ and $U_2$. The following illustrates this:

Since the $S_3$ Subsystem resource is a shared constituent of both Subsystems $U_1$ and $U_2$ the $S_3$ Subsystem has two parents: $U_1$ and $U_2$. In this case $S_3$ Subsystem has two parent Subsystems but the $S_3$ Region still must only have one parent of $S_2$ Region. This is enforced by the rule that requires all of the parents of a Subsystem to belong to the same Region. For Scoped Subsystems the Region which contains all of the Subsystem's parents is parent Region.

So far the examples have illustrated cases where the Root Subsystem has Scoped Subsystem children. It is also acceptable for an Unscoped Subsystem to be installed into the Root Region as the following figure illustrates:
134.11 Resource References

A Subsystems implementation must track the resources which are installed and determine which Subsystems reference a resource. The reference count indicates the number of installed Subsystems which reference an installed resource. The resource references and reference counts are used by the Subsystems implementation to determine if an installed resource is eligible for garbage collection and also plays a role in determining when a resource should be started and stopped, see Starting and Stopping Resources on page 593; the term reference count is only used to illustrate these concepts. The reference count of a resource is not exposed in the API of Subsystems. The following types of resources are referenced by a Subsystem:

- **Content Resources** - These are the content resources which were installed when the Subsystem was installed, that is the resources declared in the Subsystem-Content header or from the Local Repository when the Subsystem-Content header is not specified, see Determining Content on page 579. Content Resources are considered to be implicit resources.

- **Explicit Resources** - These are constituent resources which are installed explicitly, see Explicit Resources on page 589.

- **Dependencies** - These resources provide capabilities required to satisfy requirements for a Subsystem’s content resources, see Determining Dependencies on page 584. Dependencies are considered to be implicit resources.

Accepted dependency constituents are not defined as being referenced by the Subsystem of which they are a constituent unless constituent resource is a dependency for that Subsystem. Parent Subsystems are also not defined as being referenced by a child Subsystem.

134.11.1 Reference Count

When a Subsystem is being installed the Subsystems implementation must determine what resources are referenced by the Subsystem being installed. Each resource that is referenced by the Subsystem being installed will have its reference count incremented by 1. A top-level Subsystem being installed may contain child Subsystems. Each resource that is referenced by the child Subsystem being installed will have its reference count incremented by 1.

When a Subsystem is being uninstalled the Subsystems implementation must determine what resources are referenced by the Subsystem being uninstalled. Each resource that is referenced by the Subsystem being uninstalled will have its reference count decremented by 1. A top-level Subsystem being uninstalled may contain child Subsystems. Each resource that is referenced by each child Subsystem being uninstalled will have its reference count decremented by 1.
When a reference count gets set to zero then the resource is eligible for garbage collection and will be uninstalled. A Subsystems implementation may perform the garbage collection immediately or postpone the garbage collection for later. If garbage collection is postponed then the Subsystems implementation must prevent any additional usage of capabilities provided by the resource which is to be uninstalled. The garbage collection must occur in a reasonable period of time.

Bundle resources (including fragments) and Subsystem resources may be uninstalled explicitly. For example, uninstalling a Subsystem resource through the Subsystem service, or by other means outside of the Subsystems API such as uninstalling a bundle using a Bundle object. Each of the following must occur when a resource is explicitly uninstalled:

- If the resource being explicitly uninstalled was not itself installed explicitly then an error must be logged indicating that the explicitly uninstalled resource still has one or more Subsystems referencing it.
- If the resource being explicitly uninstalled was itself installed explicitly and the reference count is greater than 1 then an error must be logged indicating that the explicitly uninstalled resource still has one or more Subsystems referencing it.
- The resource being explicitly uninstalled has its reference count set to 0 and finally the resource is uninstalled.

### 134.12 Starting and Stopping Resources

A Subsystem can be started by calling the Subsystem `start` method or the Subsystems implementation can automatically start the Subsystem if the Subsystem is `ready` and the autostart setting of the Subsystem indicates that it must be started.

A Subsystem is `ready` if the Subsystem's parent is in the process of starting or is active. A started Subsystem may need to be automatically started again by the Subsystems implementation after a restart. The Subsystems implementation therefore maintains a persistent autostart setting for each Subsystem. This autostart setting can have the following values:

- **Stopped**: The Subsystem should not be started.
- **Started**: The Subsystem must be started once it is ready.

Subsystem resources which are installed as content resources, see Discovering Content Resources on page 583, of one or more Subsystems must have their autostart setting set to `started`.

When a Subsystem is started and stopped then the resources the Subsystem references may be started and stopped. See for details Starting on page 601 and Stopping on page 601.

The Subsystems implementation must track the resources which are installed and be able to determine when a resource must be started and stopped. To describe this behavior the term active use count is used. A active use count indicates the number of active Subsystems which reference a resource. The active use count is used by the Subsystems implementation in order to determine when a resource is started and stopped. The term active use count is only used to illustrate the starting and stopping of resources. The active use count of a resource is not exposed in the API of Subsystems.

Resource starting and stopping only applies to resources for which it is valid to start and stop. For example, it is not valid to start or stop resources of type osgi.fragment and a Subsystems implementation must not attempt to start or stop such resources.

### 134.12.1 Start Order

A Subsystem's Subsystem-Content header, see Subsystem-Content Header on page 579, can use the optional start-order directive for each content resource it declares. The start-order directive specifies the start order of the content resource in relation to other content resources of the Subsystem. Content resources are started in ascending order and stopped in descending order according to the
start-order directive values. Content resources with the same start-order value may be started and stopped in any order in relation to each other. There is no default value for start-order. If the start-order is not specified then a Subsystem implementation is free to start the resource in any order. For example, the following header specifies four Subsystem content resources:

Subsystem-SymbolicName: S1
Subsystem-Type: osgi.subsystem.composite
Subsystem-Content:
    A;
    type=osgi.bundle;
    version=1.0;
    start-order:=3,
    B;
    type=osgi.bundle;
    version=1.0;
    start-order:=2,
    C;
    type=osgi.bundle;
    version=1.0;
    start-order:=1,
    D;
    type=osgi.bundle;
    version=1.0;
    start-order:=2

The above headers specify a Subsystem S1 with four content resources A, B, C and D. The start-order directive is used to sort the content resources to determine the order to start or stop them when the Subsystem is started or stopped. The content resources are sorted from lowest to highest start-order. Content resources that have the same start-order value may be started and stopped in any order in relation to each other. In this example the content resources are sorted into the list [C], [B, D], [A]. C has the lowest start-order, therefore it is the first in the list. B and D have the same start-order and therefore can be started in any order in relation to each other. Finally A is last in the list because it has the highest start-order.

When the Subsystem S1 is started the content resource C must be started first, followed by the starting of B and D in any order, finally resource A is started last. When the Subsystem S1 is stopped the content resource A must be stopped first, followed by the stopping of B and D in any order, finally resource C is stopped last.

Resources that do not specify a start-order can be started and stopped in any order.

134.12.2 Active Use Count

When a Subsystem is being started the Subsystems implementation must increment the active use count of every resource which is referenced by the Subsystem being started, see Resource References on page 592. After incrementing the active use counts of the resources referenced by a Subsystem, the Subsystems implementation must determine which referenced resources need to be started. For each resource referenced by the Subsystem which is valid to be started; if the active use count is greater than zero and the resource is not currently active then the resource must be started. The collection of dependencies are started before the Subsystem's content resources. The start order for the individual resources contained in the collection of dependencies is not specified. See Start Order on page 593.

When a Subsystem is being stopped the Subsystems implementation must decrement the active use count of every resource which is referenced by the Subsystem being stopped. After decrementing the active use counts of the resources referenced by a Subsystem, the Subsystems implementation must determine which referenced resources need to be stopped. For each resource referenced by the Subsystem which is valid to be started; if the active use count equals zero and the resource is
currently active then the resource must be stopped. The Subsystem content resources are stopped before the collection of dependencies. *Start Order* on page 593 describes the stop order of the Subsystem's content resources. The stop order of the individual resources contained in the collection of dependencies is not specified.

When starting the resource types supported by this specification the following rules apply:

- `osgi.bundle` - The bundle must be transiently started using the activation policy of the bundle, that is with the Bundle.START_ACTIVATION_POLICY.
- `osgi.fragment` - Fragments cannot be started, this is a no-op.
- `osgi.subsystem.application`, `osgi.subsystem.composite`, `osgi.subsystem.feature` - The Subsystem must be transiently started if its *autostart setting* is set to *started*.

When stopping the resource types supported by this specification the following rules apply:

- `osgi.bundle` - The bundle must be persistently stopped.
- `osgi.fragment` - Fragments cannot be stopped, this is a no-op.
- `osgi.subsystem.application`, `osgi.subsystem.composite`, `osgi.subsystem.feature` - The Subsystem must be transiently stopped. Its *autostart setting* must not be changed.

Note that for resources referenced by a stopped Subsystem; bundle resources are persistently stopped and Subsystem resources are transiently stopped. This is a safeguard to handle cases where a constituent bundle is eagerly started by some other agent outside of the Subsystems implementation. Persistently started bundles will get auto started by the framework according to the start-level of the bundle. This can cause a constituent bundle to be stopped even though the Subsystem it is a constituent of is not active. To avoid this situation the Subsystems implementation always clears the persistent autostart setting of the bundle resources.

Subsystem resources which are referenced by a Subsystem are started or stopped transiently. There is no API to transiently start or stop a Subsystem. The Subsystems implementation must perform the starting or stopping of a referenced Subsystem normally except the starting or stopping process does not change the *autostart setting* for the referenced Subsystem.

### 134.13 Subsystem Service

The Subsystem service represents an Subsystem Archive resource that is installed in an OSGi Framework. The installation of a Subsystem can only be performed by using the Subsystem service API or through implementation specific means. The Subsystem interface's methods and service properties can be divided into the following categories:

- **Information** - Access to information about the Subsystem itself as well as other Subsystems that are installed.
- **Life Cycle** - The possibility to install other Subsystems and start, stop, and uninstall Subsystems.

For each Subsystem installed, the Subsystems implementation must register an associated Subsystem object as a service. The Subsystem service is used for monitoring the state of the Subsystem, for controlling the life cycle of the installed Subsystem and for installing child Subsystems.

### 134.13.1 Root Subsystem

A Subsystems implementation must register the Root Subsystem service. When a Subsystems implementation is started for the first time it must establish the Root Subsystem. The following steps are required to establish the Root Subsystem.

1. The Root Subsystem has a symbolic name `org.osgi.service.subsystem.root`, version 1.0.0 (the version of the Subsystems specification), a Subsystem id of 0 and a location of
2. The Root Subsystem has no parent Subsystem. More specifically the Root Subsystem is the only
source vertex in the Subsystem graph.
3. The Root Subsystem is considered a Scoped Subsystem of type application, with provision-poli-
cy of acceptDependencies. Since the Root Subsystem has no parent it does not import or export
any capabilities.
4. The Subsystem content is the set of bundles installed in the framework that do not belong to
any other Subsystem.
5. The root subsystem has a region context bundle as described in Region Context Bundle on page
588.

The Root Subsystem always exists when a Subsystems implementation is present and active, even
if no other Subsystems are installed. The Root Subsystem is used as the starting point for installing
Subsystems as child Subsystems. The Root Subsystem cannot be stopped or uninstalled by calling
the Subsystem service stop or uninstall methods. Any attempt to do so must result in a Subsystem
Exception.

134.13.2 Subsystem Service Properties

The primary means of discovering and monitoring a Subsystem is the Subsystem service. A Subsys-
tems implementation must register one Subsystem service for each Subsystem installed. The Sub-
system service is used for monitoring and controlling the life-cycle of the installed Subsystem. Ser-
vice properties on the Subsystem service carry most of the information required to monitor Subsys-
tem life cycle operations and the current state of a Subsystem. The following table describes the ser-
vice properties of a Subsystem:

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subsystem.id</td>
<td>Long</td>
<td>The Subsystem id of the Subsystem</td>
</tr>
<tr>
<td>subsystem.symbolicName</td>
<td>String</td>
<td>The symbolic name of the Subsystem</td>
</tr>
<tr>
<td>subsystem.version</td>
<td>Version</td>
<td>The version of the Subsystem</td>
</tr>
<tr>
<td>subsystem.type</td>
<td>String</td>
<td>The type of Subsystem.</td>
</tr>
<tr>
<td>subsystem.state</td>
<td>Subsystem.State</td>
<td>Contains the current state of the Subsystem</td>
</tr>
</tbody>
</table>

134.13.3 Subsystem States

The Subsystem service property subsystem.state contains the current state of the Subsystem (this is
referred to as the subsystem state). All Subsystem states are defined by the Subsystem.State enum, for
example, INSTALLED. The possible values of a subsystem.state are shown in the table below:

<table>
<thead>
<tr>
<th>subsystem.state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALLING</td>
<td>When a Subsystem is first installed the Subsystems implementation must register a Subsystem service with the initial subsystem.state of INSTALLING. The subsystem.state must remain in the INSTALLING state until all of the Subsystem constituents are installed successfully.</td>
</tr>
<tr>
<td>INSTALLED</td>
<td>When all contents of a Subsystem has been successfully provisioned then the subsystem.state is set to INSTALLED.</td>
</tr>
<tr>
<td>INSTALL_FAILED</td>
<td>Indicates that some failure occurred while attempting to install the Subsystem's contents.</td>
</tr>
</tbody>
</table>
subsystem.state | Description
--- | ---
RESOLVING | Starting a Subsystem triggers the resolution of a Subsystem if the subsystem.state is INSTALLED. A RESOLVING state indicates that a resolve process is occurring in an attempt to resolve all of the subsystem's content resources.
RESOLVED | Indicates that the Subsystem is resolved. A Subsystem is resolved if all of its content resources are resolved.
STARTING | Indicates that the Subsystem is in the process of being started. During this state the resources the Subsystem references which are eligible for starting are started, see Starting and Stopping Resources on page 593. Once all the eligible resources are successfully started then the subsystem.state is set to ACTIVE.
ACTIVE | The ACTIVE state indicates that all eligible resources referenced by the Subsystem were successfully started during the starting process.
STOPPING | Indicates that the Subsystem is in the process of being stopped. During this state the resources referenced by the Subsystem are stopped if appropriate.
UNINSTALLING | Indicates that the Subsystem is in the process of being uninstalled. During this state the resources referenced by the Subsystem are marked for garbage collection if they are eligible, see Resource References on page 592.
UNINSTALLED | When all of the resources referenced by the Subsystem which are eligible for garbage collection have been uninstalled then the subsystem.state is set to UNINSTALLED.

134.13.4 Subsystem Service Registrations

The Subsystems implementation must register one Subsystem service for each Subsystem installed. The Subsystems implementation must provide access to every Subsystem service from the Root Region. Every other Region must have access to the following Subsystem Services:

- Subsystem service representing the Scoped Subsystem of the Region.
- All Unscoped Subsystem services contained in the Region.
- All Subsystems which are children of a Subsystem contained in the Region.

A Region is granted access to the Subsystem services listed above automatically by the Subsystems implementation regardless of the sharing policy defined by the Scoped Subsystem of that Region. Additional Subsystem services may be imported into a Region from its parent Region by the sharing policy defined by the Scoped Subsystem of that Region.

For example, a Root Subsystem and Root Region that has two Scoped Subsystem children, S1 and S2. All Subsystem services are registered by the Subsystems implementation and are visible in the Root Region. The S1 Subsystem service is also implicitly visible in the S1 Region because it represents the Scoped Subsystem S1 contained in that Region. Similarly the S2 Subsystem service is also implicitly visible from the S2 Region. This example is depicted in Figure 134.20.

Figure 134.20 Root, attached to Scoped Subsystems S1, S2

![Diagram of Root and Scoped Subsystems]

Figure 134.21 defines a more complicated scenario where Subsystems and multiple children are involved.
All Subsystem services are visible in the Root Region. The S1 Subsystem service is also implicitly visible in the S1 Region because it represents the Scoped Subsystem S1 contained in that Region. The S1 Region also has visibility to the U1 and U2 Subsystem services because these Unscoped Subsystems are contained in the S1 Region. Similarly the S2 Subsystem service is also implicitly visible from the S2 Region. The S2 Region also has visibility to the U3 and U4 Subsystem services because these Unscoped Subsystems are contained in the S2 Region. The S2 Region also has visibility to the S3 Subsystem service because the S3 Subsystem is a child of a Subsystem contained in the S2 Region. Finally, the S3 Region has implicit visibility to the S3 Subsystem service and it has visibility to the U5 Subsystem service because the Unscoped Subsystem is contained in the S3 Region.

Note that a Scoped Subsystem’s import sharing policy may grant its Region visibility to additional Subsystem services.

### 134.13.5 Subsystem Manifest Headers

The Subsystem service interface has the `getSubsystemHeaders(Locale)` method which returns the values of the Subsystem's manifest headers. The headers returned by this method includes the values specified in the Subsystem manifest file and the values derived by the Subsystems implementation. Certain manifest headers may be derived at install time by the Subsystems implementation if they were not specified in the Subsystem manifest file. When a Subsystem manifest value is derived then the derived value must be included in the headers returned by the method `getSubsystemHeaders(Locale)`. The following Subsystem manifest headers may be derived by the Subsystems implementation:

- **Subsystem-SymbolicName**
- **Subsystem-Version**
- **Subsystem-Content**

### 134.14 Subsystem Life Cycle

The Subsystems specification provides an API to control the life cycle operations of a Subsystem. For each Subsystem installed there is an associated Subsystem object (also registered as a Subsystem service). A Subsystem's life-cycle is controlled by operations performed on the Subsystem object. Operations performed on the Subsystem may also cause equivalent operations on the resources referenced by the Subsystem. For example starting a Subsystem will cause all of its content resources to start if appropriate.
For Scoped Subsystems the export and import sharing policies are initially disabled at runtime and get enabled at runtime by the Subsystems implementation depending on the state of the Scoped Subsystem which defines the sharing policy. When an import sharing policy is disabled at runtime, none of the installed resources contained in the Region associated with the Scoped Subsystem have visibility to capabilities available in the parent Region. Once an import policy is enabled at runtime the installed resources contained in the Region have visibility to capabilities available in the parent Region according to what the import sharing policy specifies. When an export sharing policy is disabled at runtime, none of the capabilities provided by installed resources contained in the Region associated with the Scoped Subsystem are visible in the parent Region. Once an export policy is enabled at runtime the capabilities provided by installed resources contained in the Region are visible in the parent Region according to what the export sharing policy specifies.

The subsystem.state is a reflection of the last action performed on the Subsystem through the Subsystem service. The use of any other API to change the state of a resource referenced by a Subsystem directly does not result in a change of the subsystem.state (i.e. calling stop on a bundle). For example, uninstalling a Subsystem content resource which is a bundle does not cause the Subsystem to be uninstalled, but it does result in an error being logged.

All references to changing the state of this Subsystem include both changing the state of the Subsystem object as well as the state property of the Subsystem service.

The following figure illustrates the life cycle of a Subsystem:

**State diagram Subsystems**

![State diagram Subsystems](image)

### 134.14.1 Installing

A Subsystem's install process is initiated using one of the Subsystem service's install methods. The Subsystems implementation must assign a unique Subsystem identifier that is higher than any previous installed Subsystem identifier. Previously installed Subsystem identifiers include Subsystems which were uninstalled in a previous session of the framework. The installation of a Subsystem must be:
- *Persistent* - The Subsystem must remain installed across framework and Java VM invocations until the Subsystem is explicitly uninstalled.
- *Atomic* - The install method must completely install the Subsystem or, if installation fails, the Subsystems implementation must leave the framework in the same state as it was before the method was called.

Once a Subsystem has been installed, a Subsystem object is created and all remaining life cycle operations for the installed Subsystem must be performed upon this object. The returned Subsystem object can be used to start, stop, and uninstall the Subsystem as well as install child Subsystems.

When a Subsystem is being installed the Subsystems implementation must perform the following operations synchronously before returning from the install method:

1. Determine the symbolic name, version, and type for the Subsystem being installed as defined in *Subsystems* on page 568. If the Subsystem name, version or type are invalid then the install fails and a Subsystem Exception is thrown.
2. Determine the Subsystems for which the Subsystem being installed will become a constituent of by following the steps in *Explicit and Implicit Resources* on page 588.
3. Determine the Subsystem identifier. Subsystem identifiers are unique and assigned by the Subsystems implementation.
4. If the Subsystem is a Scoped Subsystem then create the new Region for the Subsystem and install and start the Region context bundle. See *Region Context Bundle* on page 588 for the Region context bundle.
5. Register a Subsystem service with the initial `subsystem.state` service property set to `INSTALLING`. This Subsystem service represents the Subsystem resource. See *Subsystem Service Properties* on page 596 and *Subsystem Service Registrations* on page 597 for more details.
6. Determine the Subsystem content resources. See *Determining Content* on page 579 for details on how the Subsystem contents are determined. If the contents cannot be discovered successfully and the content is not optional then an installation failure occurs and a Subsystem Exception is thrown. Otherwise continue to the next step.
7. Determine the Subsystem dependencies. See *Determining Dependencies* on page 584 for details on determining the Subsystems dependencies. If the dependencies cannot be determined successfully then an installation failure occurs and a Subsystem Exception is thrown. Otherwise continue to the next step.
8. Install the dependencies. The Subsystems implementation must prevent resolution of dependency wires to the capabilities provided by the installed dependencies until the Subsystem has successfully entered `INSTALLED` state. See *Explicit and Implicit Resources* on page 588 for details on where dependencies are installed and see *Resource References* on page 592 for how they are tracked. If any dependency fails to install then an installation failure occurs and a Subsystem Exception is thrown. Otherwise continue to the next step.
9. Install content resources. The content resources must be disabled from resolving until the Subsystem has successfully entered `INSTALLED` state. If any content resource fails to install then an installation failure occurs and a Subsystem Exception is thrown. Otherwise continue to the next step.
10. If the Subsystem is scoped, enable the import sharing policy for the Region. See *Sharing Capabilities* on page 587.
11. Enable resolution for all of the Subsystem content and any dependencies installed. Set the `subsystem.state` to `INSTALLED` and return the installed Subsystem object.

The state `INSTALL_FAILED` is used to inform about an installation failure. All installation failures use the following steps:

1. When a Subsystem fails to install it enters the `INSTALL_FAILED` state.
2. Immediately transition the Subsystem to the `UNINSTALLING` state.
3. All content and dependencies which may have been installed by the Subsystem installing process must be uninstalled.
4. Transition the Subsystem to the UNINSTALLED state.
5. Unregister the Subsystem service.
6. If the Subsystem is scoped then, uninstall the Region context bundle.
7. Throw a Subsystem Exception indicating an install failure.

134.14.2 Resolving

A Subsystem's resolve process is initiated by performing a start operation on a Subsystem whose subsystem.state is currently set to INSTALLED. There is no explicit operation for initiating the resolve process of a Subsystem. The Subsystems implementation is free to initiate the resolve process for a Subsystem for any reason. For example, the Subsystems implementation may choose to try to resolve all currently installed Subsystems when the start operation is performed on a single Subsystem.

134.14.3 Starting

A Subsystem can be started by calling the Subsystem start() method or the Subsystems implementation can automatically start the Subsystem if the Subsystem is ready and the autostart setting of the Subsystem indicates that it must be started. When a Subsystem is being started the Subsystems implementation must perform the following operations synchronously before returning from the start() method:

1. If the subsystem.state is INSTALL_FAILED, UNINSTALLED, or UNINSTALLING, then an Illegal State Exception is thrown.
2. Set the Subsystems autostart setting to started.
3. If the subsystem.state is ACTIVE then the start method returns immediately.
4. If the Subsystem is not ready to be started then the start method returns immediately.
5. If this subsystem.state is RESOLVING, STARTING, or STOPPING, then the start method must wait for starting or stopping to complete before continuing. If this does not occur in a reasonable time, a Subsystem Exception is thrown to indicate the Subsystem was unable to be started.
6. If the subsystem.state is RESOLVED then continue to the next step; otherwise if the subsystem.state is INSTALLED then the subsystem.state is set to RESOLVING and an attempt is made to resolve all of the Subsystem's content resources. If all contents are resolved then set the subsystem.state to RESOLVED, enable the export sharing policy and continue to the next step; otherwise a starting failure occurs and a Subsystem Exception is thrown.
7. Set the subsystem.state to STARTING.
8. Start all resources referenced by the Subsystem according to Starting and Stopping Resources on page 593. If all of the resources start successfully then continue to the next step; otherwise a start failure occurs.
9. Set the subsystem.state to ACTIVE and return.

All start failures use the following steps:

1. If the subsystem.state is STARTING then change the state to STOPPING.
2. Stop all resources that were started as part of this operation.
3. Change the state to INSTALLED or RESOLVED depending on if the Subsystem was resolved.
4. Throw a Subsystem Exception indicating the cause of the start failure.

134.14.4 Stopping

A Subsystem's stop process is initiated using the Subsystem service's stop() method. When a Subsystem is being stopped the Subsystems implementation must perform the following operations synchronously before returning from the stop() method:
1. If the subsystem.state is UNINSTALLED, INSTALL_FAILED, or UNINSTALLING, then an Illegal State Exception is thrown.

2. Set the Subsystems autostart setting to stopped.

3. If the subsystem.state is RESOLVED or INSTALLED then the stop() method returns immediately.

4. If this subsystem.state is STARTING or STOPPING, then the stop method must wait for starting or stopping to complete before continuing. If this does not occur in a reasonable time, a Subsystem Exception is thrown to indicate the Subsystem was unable to be stopped.

5. Set the subsystem.state to STOPPING.

6. Stop all resources referenced by the Subsystem according to Starting and Stopping Resources on page 593. If any error occurs while stopping a resource the Subsystems implementation must continue to stop the remaining resources that are eligible to stop.

7. Set the subsystem.state to RESOLVED.

With regard to error handling while stopping resources referenced by the Subsystem, errors subsequent to the first should be logged. Once the stop process has completed, a Subsystem Exception must be thrown with the initial error as the specified cause.

134.14.5 Uninstalling

A Subsystem's uninstall process is initiated using the Subsystem service's uninstall() method. To whatever extent possible, the Subsystems implementation must determine the resources referenced by the Subsystem which are eligible for garbage collection, Reference Count on page 592. This method must always uninstall the Subsystem from the persistent storage of the Subsystems implementation.

Once this method returns, the state of the platform must be the same as if the Subsystem had never been installed, unless some bundle resource which was uninstalled has exported package which are being used by other bundles still installed in the platform. All old exports must remain available for existing bundles and future resolves until the uninstalled bundle is refreshed or the framework is restarted.

When a Subsystem is being uninstalled the Subsystems implementation must perform the following operations before returning from the uninstall() method:

1. If the subsystem.state is UNINSTALLED then this method returns immediately.

2. If the subsystem.state is STARTING, STOPPING or ACTIVE then the Subsystem is stopped according to Stopping on page 601. Otherwise continue to the next step.

3. If the subsystem.state is INSTALLING and the installing process is interruptible, fail the install process; otherwise, wait until the installation is complete.

4. If the subsystem.state is in the INSTALL_FAILED state then skip to step 6.

5. Set the subsystem.state to INSTALLED.

6. Set the subsystem.state to UNINSTALLING.

7. Determine the resources referenced by the Subsystem which are eligible for garbage collection according to Reference Count on page 592. If a Subsystems implementation does garbage collection synchronously and any error occurs while uninstalling a resource the Subsystems implementation must continue to uninstall the remaining resources that are eligible to garbage collect.

8. Set the subsystem.state to UNINSTALLED.

9. Unregister the Subsystem service.

10. If the Subsystem is a Scoped Subsystem then uninstall the Region context bundle. At this point the Region no longer exists.

With regard to error handling while synchronously uninstalling resources eligible for garbage collection, errors subsequent to the first should be logged. Once the uninstall process has completed, a Subsystem Exception must be thrown with the initial error as the specified cause.
134.15 **Pre-Calculated Deployment**

A pre-calculated deployment in the form of a deployment manifest can be included as part of a Subsystem Archive or provided by a deployer at installation time. Manifests provided at install time override those included within an archive, and those within an archive override calculated ones. The deployment manifest defines the precise deployment of the Subsystem. Providing a deployment manifest means a Subsystem can be deployed and the exact resources that are installed are known ahead of time. This allows test teams to test specific deployments and these same deployments can then be used in production. The deployment manifest is a locking down of the variability in a Subsystem manifest (or the equivalent if the Subsystem definition is calculated during deployment based on the Subsystem Archive). The deployment manifest follows the same syntax rules as the Subsystem manifest but uses different headers for deployment-specific information. A deployment manifest describes the following:

- The exact versions for content resources
- Any dependencies required to resolve the Subsystem's content that are not satisfied by the target runtime
- Sharing policy for requirements and capabilities shared into or out of the Subsystem.

Because a Deployment Manifest's dependencies bridge between the requirements of the Subsystem and the capabilities of the target runtime, it is not guaranteed to be portable. If available, the Subsystem service implementation must first attempt to use the Deployment Manifest to deploy the Subsystem. If the Deployment Manifest is found not to work, for example, the chosen resources do not resolve for the target runtime, then the Subsystem's implementation must fail the installation of the Subsystem.

### 134.15.1 Deployment Headers

A Subsystem can carry descriptive information about its deployment in the Deployment Manifest file contained in its Subsystem Archive under the name `OSGI-INF/DEPLOYMENT.MF`. This specification defines Deployment Manifest headers such as `Deployed-Content`, which Subsystem deployers (typically tools) use to supply deployment information about a Subsystem. A Subsystems implementation must:

- Process the main section of the manifest. Individual section of the manifest are ignored.
- Ignore unknown manifest headers. The Subsystem deployer can define additional manifest headers as needed.
- Ignore unknown attributes and directives.

All specified manifest headers are listed in the following sections, and include example values. All headers are optional, unless specifically indicated.

#### 134.15.1.1 Deployment-ManifestVersion: 1

The Deployment-ManifestVersion header defines that the deployment manifest follows the rules of a Subsystems Specification. It is 1 (the default) for this version of the specification. Future versions of the Subsystems Specification can define higher numbers for this header.

#### 134.15.1.2 Subsystem-SymbolicName: com.acme.subsystem.logging

The Subsystem-SymbolicName header specifies a non-localizable name for the Subsystem that the deployment manifest is for. The Subsystem symbolic name together with a version must identify a unique Subsystem though it can be installed multiple times in a framework. See *Validating Subsystem Identity* on page 604.
134.15.1.3 Subsystem-Version: 1.0

The Subsystem-Version header specifies the version of this Subsystem that the deployment manifest is for. See Validating Subsystem Identity on page 604.

134.15.1.4 Deployed-Content: com.acme.logging;type=osgi.bundle;deployed-version=1.0.0

The Deployed-Content header lists requirements for the exact resources that are considered to be the contents of this Subsystem. This header identifies the exact versions of the resources listed in the Subsystem-Content header. See Deployed-Content on page 605.

134.15.1.5 Provision-Resource: com.acme.logging;type=osgi.bundle;deployed-version=1.0.0

The Provision-Resource header lists requirements for the exact resources to be installed in order to satisfy requirements from the Deployed-Content resources that are not satisfied by the capabilities of the target runtime. See Provision-Resource on page 605.

134.15.1.6 Import-Package: com.acme.api;version="[1.0,1.1)"

The Import-Package header lists package requirements for capabilities that are to be imported into a Scoped Subsystem. See Import-Package on page 606.

134.15.1.7 Export-Package: com.acme.api;version=1.0.1

The Export-Package header lists package capabilities that are to be exported out of a Scoped Subsystem. See Export-Package on page 606.

134.15.1.8 Require-Bundle: com.acme.logging;version="[1.0,1.1)"

The Require-Bundle header lists bundle requirements for bundle capabilities that are to be imported into a Scoped Subsystem. See Require-Bundle on page 607.

134.15.1.9 Provide-Capability: com.acme.dict; from=nl; to=de; version:Version=1.2


134.15.1.10 Require-Capability: osgi.ee; filter="(osgi.ee=*)"


134.15.1.11 Subsystem-ImportService: com.acme.service.Logging

The Subsystem-ImportService header lists service requirements for service capabilities that are to be imported into a Scoped Subsystem. See Services on page 607.

134.15.1.12 Subsystem-ExportService: com.acme.service.Logging

The Subsystem-ExportService header lists service requirements that are matched against service capabilities provided by the Deployed-Content resources. Any matching capabilities are exported out of the Scoped Subsystem.

134.15.2 Validating Subsystem Identity

The Subsystem to which the deployment manifest applies is identified by the Subsystem's symbolic name and version headers. These headers are identical to those specific in the Subsystem manifest. A Subsystem runtime must validate that the headers specified in the deployment manifest match those of the Subsystem manifest, taking into account Subsystem manifest defaulting rules. This allows the two manifests to be managed by teams separately during development or testing whilst ensuring no mistakes have been made when they are brought together for deployment. If the headers do not match, then the runtime must not use the deployment manifest and must fail the installation.
134.15.3 Deployed-Content

The Deployed-Content header lists the exact constituents to be installed for the Subsystem. For each mandatory entry in the Subsystem-Content header, there must be a corresponding Deployed-Content entry. If a content resources is identified as optional and there is a corresponding entry in the deployment manifest, then it must be deployed. If there is no corresponding entry in the deployment manifest then no resource must be deployed for it. The Deployed-Content entry identifies the exact version of the constituent whereas the Subsystem-Content entry may specify a version range. Each Deployed-Content entry is identified by symbolic name, version and type (an osgi identity).

Deployed-Content:
- com.acme.logging;
  deployed-version=1.0,
- com.acme.persistence;
  deployed-version=1.1;
  type=osgi.subsystem.composite

Each entry must uniquely identify the resource to be provisioned as a constituent of the Subsystem. The following mandatory matching attributes must be applied to each entry:

- **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. There is no default value for this attribute.

The following architected matching attribute as well as any arbitrary matching attributes can be applied to each entry:

- **type** - The type of the constituent. 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. bundles). The default value is osgi.bundle. A Subsystems implementation may support additional types, but the following types must be supported:
  - osgi.bundle
  - osgi.fragment
  - osgi.subsystem.application
  - osgi.subsystem.composite
  - osgi.subsystem.feature

The value of this directive must match the type directive for the corresponding entry in the Subsystem-Content header, including taking into account defaulting. If the type does not match, then the installation must fail.

The following directive can be applied to each entry:

- **start-order** - The precedence the constituent should have during the start sequence. Resources with lower start-order values are started before resources with higher values. Resources with the same start-order value may be started sequentially or in parallel. The value of this directive must match the start-order directive for the corresponding entry in the Subsystem-Content header, including taking into account defaulting.

134.15.4 Provision-Resource

The Provision-Resource header lists the resources to be provisioned in support of the Subsystem's dependencies. The exact location in the Subsystem hierarchy where the resources are installed is determined by the provision-policy of the Subsystem or its parents.
The Provision-Resource header must result in a transitively complete deployment. For example, if a resource added to Provision-Resource brings in additional unsatisfied requirements, further resources must be added to satisfy these, until there are no unresolved requirements remaining.

Provision resource 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. There is no default value for this attribute.

The following architected matching attributes as well as any arbitrary matching attributes can be applied to each entry:

- `type` - The type of the 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., bundles). The default type is `osgi.bundle`. A Subsystems implementation may support additional types, but the following types must be supported:
  - `osgi.bundle`
  - `osgi.fragment`
  - `osgi.subsystem.application`
  - `osgi.subsystem.composite`
  - `osgi.subsystem.feature`

The list of the Provision-Resource entries is determined by resolving the Subsystem's requirements. The way in which the Subsystem's requirements are resolved is dependent on the Subsystem's sharing policy.

For a Scoped Subsystem the provision resources header must identify a set of resources necessary to satisfy the requirements into the Subsystem that are not satisfied by the target deployment environment. These requirements may be for packages, services, or other types of requirements, and are those identified in the deployment manifest using headers such as `Import-Package` and `Subsystem-ImportService`.

For an Unscoped Subsystem any mandatory requirements that are not satisfied by capabilities provided by the target environment may be satisfied by other constituents or a resource added to the Provision-Resource header. The resolution process for Unscoped Subsystems has no propensity to resolve to capabilities provided by the Subsystem's constituents and so a resource listed in Provision-Resource may provide capabilities that are also provided by a constituent resource.

### 134.15.5 Import-Package

Scoped Subsystems describe the exact packages they import in their Deployment Manifests. They do this using the bundle `Import-Package` header. Any packages that match the `Import-Package` statement must be allowed into the Scoped Subsystem by its associated Region's sharing policy.

Unscoped Subsystems have a sharing policy that shares all packages and therefore their deployment manifests do not use this header to describe the sharing of individual packages. If this header is present and the Subsystem is unscoped, then the runtime must fail the installation of the Subsystem.

### 134.15.6 Export-Package

Scoped Subsystems describe the exact packages they export in their deployment manifests. They do this using the bundle `Export-Package` header. Any packages that match the `Export-Package` statement must be made available outside the Subsystem by its associated Region's sharing policy.

Unscoped Subsystems have a sharing policy that shares all packages and therefore their deployment manifests do not use this header to describe the sharing of individual packages. If this header
is present and the Subsystem is unscoped, then the runtime must fail the installation of the Subsystem.

### 134.15.7 Require-Bundle

Scoped Subsystems can have Require-Bundle requirements satisfied by bundles outside the Subsystem. These bundle requirements are described using the bundle Require-Bundle header. Any bundles that match the Require-Bundle statement must be allowed into the Scoped Subsystem by its associated Region's sharing policy. If a bundle matches the Require-Bundle requirement then it becomes available as a candidate for wiring any Require-Bundle requirements inside the Subsystem. However, any packages the matching bundle provides are not made available to satisfy Import-Package requirements by the Region's sharing policy. If the packages are also required then they must be listed in the deployment manifest's Import-Package header.

Unscoped Subsystems have a sharing policy that shares all bundles and therefore their deployment manifests do not use this header to describe the sharing of specific bundles. If this header is present and the Subsystem is unscoped, then the runtime must fail the installation of the Subsystem.

### 134.15.8 Services

Scoped Subsystems can import and export services using the Subsystem-ImportService and Subsystem-ExportService headers respectively. These two headers must conform to the following syntax:

```text
Subsystem-ImportService ::= service( ',' service )*
Subsystem-ExportService ::= service ( ',' service )*

service ::= qname ( ';' parameter )*
```

Both headers support the following directive:

- filter - A filter expression that is used to match against the service properties of services registered using the specified qname of the service's object class. The filter directive is optional. If no filter directive is defined then all services registered using the specified qname match the service statement.

### 134.15.9 Subsystem-ImportService

Scoped Subsystems describe the services they import in their deployment manifests. They do this using the Subsystem-ImportService header. Subsystem-ImportService header defines a list of OSGi service filters that are matched against the services visible inside the Scoped Subsystem's parent Region. Each service visible in the Subsystem's parent Region that matches one or more Subsystem-ImportService statements must be allowed into the Scoped Subsystem by its associated Region's sharing policy. The following example imports services registered under the com.acme.logging.Log interface with a service property threshold=error.

```text
Subsystem-ImportService: com.acme.logging.Log;filter="(threshold=error)"
```

Unscoped Subsystems have a sharing policy that shares all services and therefore their deployment manifests do not use this header to describe the sharing of specific services. If this header is present and the Subsystem is unscoped, then the runtime must fail the installation of the Subsystem.

### 134.15.10 Subsystem-ExportService

Scoped Subsystems describe the services they export in their deployment manifests. They do this using the Subsystem-ExportService header. The Subsystem-ExportService header defines a list of OSGi service filters that are matched against the services visible inside the Scoped Subsystem's Region. Each service visible in the Scoped Subsystem's Region that matches one or more Subsystem-ExportService statements must be allowed by its associated Region's sharing policy into the Scoped Subsystem's parent Region. The following example exports services registered under the com.acme.logging.Log interface with a service property threshold=error.

```text
Subsystem-ExportService: com.acme.logging.Log
```
Subsystem Types

Subsystem types simplify the configuration of sharing policies. The type of Subsystem is specified using the Subsystem-Type header. Each type has its own default sharing policy, for example, to forbid the sharing of capabilities out, or to share all capabilities in. This specification defines three Subsystem types:

- osgi.subsystem.application
- osgi.subsystem.composite
- osgi.subsystem.feature

Other, non-standard, types are permitted. The specifics of each standard type are describe below.

134.16 Application

An application is a Scoped Subsystem with a sharing policy associated with what is often considered to be an application. An application does not share (export) any capabilities to other bundles or Subsystems. It also does not explicitly import any capabilities. Any required capabilities that are not satisfied by the application's constituents are automatically shared in (imported) from the parent Subsystem.

A Subsystem is identified as an application by specifying a Subsystem type value of osgi.subsystem.application in the Subsystem manifest.

Subsystem-Type: osgi.subsystem.application

134.16.2 Application Deployment

Application Subsystems are not configured using additional requirement or capability headers, such as Import-Package. Applications do not export any capabilities. If an application Subsystem contains any capability exports then the Subsystem runtime should log an error and must fail.

Any imported capabilities are derived from the application Subsystem content. An application Subsystem implicitly imports any capabilities required to satisfy requirements from the Subsystem contents that are not satisfied by the capabilities of the Subsystem content.

Unsatisfied mandatory requirements result in a subsystem installation failure. Unsatisfied optional requirements do not. However, implementations must ensure any unsatisfied optional requirements are added to the sharing policy.

134.16.2.1 Package Imports

Application resolution is required to prefer packages provided by content bundles over those provided outside the application. For this reason, the application Subsystem sharing policy only imports packages corresponding to Import-Package statements from the content bundles that are not satisfied when resolving the application contents in isolation. This is equivalent to first resolving the Subsystem-Content requirements to determine the Deployed-Content and then based on this set of resources, determining which Import-Package requirements remain unsatisfied.

A deployment manifest for an application Subsystem would list these package imports using the Import-Package header.
134.16.2.2 Service Imports

Application resolution is required to prefer services provided by content bundles over those provided outside the application. For this reason, the application Subsystem sharing policy only imports services required by the Subsystem's content bundles that are not also provided by the content bundles. This specification provides a means of declaratively identifying the services a bundle provides or requires using the Provide-Capability and Require-Capability headers with the osgi.service namespace. See osgi.service Namespace on page 636

An example of a bundle providing the service and declaring it using the Provide-Capability header is as follows:

```
Provide-Capability: osgi.service;
    objectClass=com.foo.MyService;
    a.service.prop=somePropertyValue
```

Note that declaring a provided service in this manner only affects resolution. It does not affect service visibility at runtime. In other words, a subsystem that imports service com.acme.Foo will see all of the corresponding service registrations that its parent sees regardless of whether or not the provider declared this service in the Provide-Capability header.

An example of a bundle requiring a service and declaring the requirement using the Require-Capability header is as follows:

```
Require-Capability: osgi.service;
    filter:="(&(objectClass=com.foo.MyService)(a.service.prop=somePropertyValue))"
```

These headers can be hand-written (e.g., to declare programmatic use of an OSGi service) or generated by a tool (e.g., BND) based on a declarative component model configuration (e.g., Declarative Services or Blueprint). A Subsystems implementation must assume these headers, if present, declare all of the service dependencies. Implementations must therefore not search the bundle for additional dependencies from other sources.

A deployment manifest for an application Subsystem would list these service imports using the Subsystem-ImportService header.

134.16.2.3 Bundle Requirements

Application resolution is required to prefer bundle capabilities provided by content bundles over those provided outside the application. For this reason, the application Subsystem sharing policy only requires bundle capabilities corresponding to Require-Bundle statements from the content bundles that are not satisfied when resolving the application contents in isolation. This is equivalent to first resolving the Subsystem-Content requirements to determine the Deployed-Content and then based on this set of resources, determining which Require-Bundle requirements remain unsatisfied.

A Deployment Manifest for an application Subsystem would list these bundle requirements using the Require-Bundle header.

134.16.2.4 Generic Requirements

Application resolution is required to prefer generic capabilities provided by content bundles over those provided outside the application. For this reason, the application Subsystem sharing policy only generic requirements corresponding to Require-Capability statements from the content bundles that are not satisfied by Provide-Capability statements of the content bundles when resolving the application contents in isolation. This is equivalent to first resolving the Subsystem-Content requirements to determine the Deployed-Content and then based on this set of resources, determining which Require-Capability statements remain unsatisfied.

An deployment manifest for an application Subsystem would list these generic requirements using the Require-Capability header.
134.16.2.5 Dependencies

Application Subsystems’ implicit requirements are determined as described in the Application Deployment section in Determining Dependencies on page 584. Any mandatory requirements from constituents that are not satisfied by capabilities provided by the target environment or other constituents must be satisfied by additional dependencies. The Subsystem runtime is responsible for provisioning these based on the Subsystem's provision policy or those of its scoped parents. If the application Subsystem has an associated deployment manifest, then these resources are described in the Provision-Resource header.

134.16.3 Composite

A composite is a Scoped Subsystem with a sharing policy that by default does not share anything with its parent and therefore all sharing is fully explicit. Capabilities, such as packages and services, may be explicitly imported into or exported out of the composite.

A Subsystem is identified as a composite by specifying a Subsystem type value of osgi.subsystem.composite in the Subsystem manifest.

Subsystem-Type: osgi.subsystem.composite

134.16.3.1 Subsystem Content

The Subsystem-Content header allows version ranges for content resources. For composite Subsystems, this value must be a fixed version range (e.g. [1.0, 1.0]) for resources of type osgi.bundle, osgi.fragment, osgi.subsystem.application, osgi.subsystem.composite, and osgi.subsystem.feature. This is due to the fact that there is an inextricable link between the versions on the explicit import and export statements made on a composite and the chosen versions of the content bundles. Allowing variability in the content versions for these types of resources risks introducing incompatibilities with sharing policy for the composite. If a composite Subsystem does not use strict version ranges then the composite Subsystem must fail to install.

134.16.3.2 Package Imports

A composite Subsystem explicitly states the packages it imports using the Import-Package header. If the composite includes a deployment manifest then the Import-Package header is used to describe these and they must be identical (logically, not syntactically) to the Import-Package headers in the composite's Subsystem manifest. If the imports are not the same then the Subsystem runtime should log an error and must fail the installation.

134.16.3.3 Package Exports

A composite Subsystem explicitly states the packages it exports using the Export-Package header. If the composite includes a deployment manifest then the Export-Package header is used to describe these and they must be identical (logically, not syntactically) to the Export-Package headers in the composite's Subsystem manifest. If the exports are not the same then the Subsystem runtime should log an error and must fail the installation.

134.16.3.4 Service Imports

A composite Subsystem explicitly states the services it imports using the Subsystem-ImportService header (see Subsystem-ImportService on page 607). For example:

Subsystem-ImportService: com.acme.logging.Log

If the composite includes a deployment manifest then the Subsystem-ImportService header is used to describe these and they must be identical (logically, not syntactically) to the Subsystem-ImportService headers in the composite's Subsystem manifest. If the imports are not the same then the Subsystem runtime should log an error and must fail the installation.
134.16.3.5 Service Exports

A composite Subsystem explicitly states the services it exports using the Subsystem-ExportService header (see Subsystem-ExportService on page 607). For example:

Subsystem-ServiceExport: com.acme.logging.Log

If the composite includes a deployment manifest then the Subsystem-ExportService header is used to describe these and they must be identical (logically, not syntactically) to the Subsystem-ExportService headers in the composite's Subsystem manifest. If the exports are not the same then the Subsystem runtime should log an error and must fail the installation.

134.16.3.6 Bundle Requirements

A composite Subsystem explicitly states the bundles it requires using the Require-Bundle header. If the composite includes a deployment manifest then the Require-Bundle header is used to describe these and the requirements must be identical (logically, not syntactically) to the Require-Bundle requirements in the composite's Subsystem manifest. If the requirements are not the same then the Subsystem runtime should log an error and must fail the installation.

134.16.3.7 Generic Requirements

A composite Subsystem explicitly states the generic capabilities it requires using the Require-Capability header. If the composite includes a deployment manifest then the Require-Capability header is used to describe these and they must be identical (logically, not syntactically) to the Require-Capability headers in the composite's Subsystem manifest. If the capability requirements are not the same then the Subsystem runtime should log an error and must fail the installation.

134.16.3.8 Generic Capabilities

A composite Subsystem explicitly states the generic capabilities it provides using the Provide-Capability header. If the composite includes a deployment manifest then the Provide-Capability header is used to describe these and they must be identical (logically, not syntactically) to the Provide-Capability headers in the composite's Subsystem manifest. If the capabilities are not the same then the Subsystem runtime should log an error and must fail the installation.

134.16.3.9 Dependencies

A composite Subsystem's explicit requirements are stated in the Subsystem manifest. Any mandatory requirements that are not satisfied by capabilities provided by the target environment must be satisfied by additional dependencies. The Subsystem runtime is responsible for provisioning these based on the Subsystem's provision policy or the provision policy of its scoped parents. If the composite Subsystem has an associated deployment manifest, then these resources are described in the Provision-Resource header.

134.16.4 Feature

A feature is an Unscoped Subsystem and therefore provides no isolation of its own. A feature does however always exist in the context of one and only one Region which can restrict the capabilities a feature can see and the extent to which a feature's capabilities are shared.

A Subsystem is identified as a feature by specifying a Subsystem type value of osgi.subsystem.feature in the Subsystem manifest.

Subsystem-Type: osgi.subsystem.feature
134.16.4.1 Explicit Requirements and Capabilities

A feature Subsystem implicitly imports and exports all requirements and capabilities. If the feature Subsystem include any headers designed to modify the sharing policy of a Subsystem, such as Import-Package or Subsystem-ImportService, then the Subsystem runtime should log an error and must fail the installation of the Subsystem.

134.16.4.2 Dependencies

Feature Subsystems implicitly import all capabilities. A Subsystem runtime is responsible for provisioning any dependencies necessary for the Subsystem's constituents to resolve. The calculation of the dependencies can also take into account capabilities provided by the target runtime. The dependencies can include resources that provide capabilities equivalent to those provided by one or more of the constituent resources where the dependency's capability is a considered a better match in the context of some resolution. The Subsystem runtime is responsible for provisioning the dependencies based on the Subsystem's provision policy or the provision policy of its scoped parents. If the feature Subsystem has an associated deployment manifest, then these dependencies are described in the Provision-Resource header.

134.17 Weaving Hooks

Subsystems implementations must ensure that dynamic package imports added by weaving hooks are available to subsystems whose classes have been woven by updating the sharing policies.

Dynamic package imports added by weaving hooks are observed by registering a WovenClassListener service and receiving notifications via the WovenClassListener.modified(WovenClass) method. The sharing policy must be updated while the woven class is in the TRANSFORMED state so that it takes effect before the bundle wiring is updated during the transition to DEFINED; otherwise, the class would fail to load.

The bundle containing the woven class can be obtained by calling the WovenClass.getBundleWiring().getBundle() method. A bundle might be a constituent of multiple subsystems, but never more than one scoped subsystem. The rest are features, which have no sharing policies to update. It's possible the bundle will not be a constituent of a scoped subsystem. The scoped subsystem, if any, containing the bundle as a constituent is retrieved.

It's possible for a classload request to occur on a bundle in an unresolved subsystem because the framework is free to resolve bundles whenever it desires. A resolved bundle can potentially receive a classload request. For example, a BundleEventListerner registered with the system bundle context could receive the RESOLVED event and, for whatever reason, load a class. Also, a resolved bundle in an unresolved feature might get wired to another bundle. If this is the case, the subsystem must be resolved in order to guarantee the dynamic imports will not effect the resolution and, therefore, potentially create a wiring inconsistent with the deployment manifest. Just as the framework is free to resolve bundles at anytime and for whatever reason, Subsystems implementations are free to resolve subsystems.

The sharing policy is only updated if the dynamic import cannot be completely satisfied from within the subsystem. Note that all dynamic imports with a wildcard must always be added to the sharing policy.
134.18 Stopping and Uninstalling Subsystems Implementation

When the Subsystems implementation is stopped all of the installed Subsystems must be persistently stored and present when the Subsystems implementation becomes active again. This includes any bundles that got installed as part of a Subsystem installation. The Subsystems implementation is not required to do any additional cleanup when the Subsystems implementation is stopped or uninstalled. All bundles that got installed as a result of installing a Subsystem may still be installed after stopping or uninstalling the Subsystems implementation bundle. If it is important to clean up the bundles associated with a Subsystem installation then the Subsystem should should be uninstalled before uninstalling the Subsystems implementation.

134.19 Security

134.19.1 Subsystem Permission

The Subsystem Permission is a permission used to grant the right to manage Subsystems with the option to restrict this right to a subset of Subsystems, called targets. For example, an operator can give a bundle the right to only manage Subsystems with a symbolic-name prefix of com.acme:

```
...SubsystemPermission("(name=com.acme.*)",
...)
```

The actions of Subsystem Permission are fine-grained. They allow a deployer to assign only the permissions that are necessary for a bundle. For example, a bundle may be granted only the permission to start and stop all Subsystems:

```
...:SubsystemPermission[**, EXECUTE]
```

Code that needs to check Subsystem Permission must always use the constructor that takes a Subsystem as a parameter: `SubsystemPermission(Subsystem, String)` with a single action.

For example, the implementation of `Subsystem.start` method must check that the caller has access to execute the Subsystem:

```
public class SubsystemImpl implements Subsystem{
  public void start() {
    securityManager.checkPermission(new SubsystemPermission(this, "execute");
  }
}
```

The Subsystem Permission takes a Filter as its name argument. Filter based permissions are described in [1] Filter Based Permissions. Subsystem Archives are not signed and therefore the signer key is not supported. The keys have the following meaning for the Subsystem Permission:

- id - The Subsystem ID of a Subsystem. For example (id=23)
- location - The location of a Subsystem. For example (location=https://www.acme.com/download/*)
- name - The symbolic name of a Subsystem. For example (name=com.acme.*)

The name parameter of the permission can also be a single wildcard character (`*` \u002a). In that case all Subsystems must match.
### 134.19.2 Actions

The action parameter of Subsystem Permission will specify the subset of privileged Subsystem management operations that are allowed. The actions that are architected are listed below. Future versions of the specification can add additional actions. The given set should therefore not be assumed to be a closed set.

<table>
<thead>
<tr>
<th>Action</th>
<th>Used in</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTEXT</td>
<td>Subsystem.getBundleContext</td>
</tr>
<tr>
<td>METADATA</td>
<td>Subsystem.getSubsystemHeaders</td>
</tr>
<tr>
<td></td>
<td>Subsystem.getLocation</td>
</tr>
<tr>
<td>LIFECYCLE</td>
<td>Subsystem.install</td>
</tr>
<tr>
<td></td>
<td>Subsystem.uninstall</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Subsystem.start</td>
</tr>
<tr>
<td></td>
<td>Subsystem.stop</td>
</tr>
</tbody>
</table>

### 134.19.3 Required Permissions

A Subsystems implementation must check the caller for the appropriate Subsystem Permission before initiating a Subsystem management operation (e.g. install, start, stop, uninstall). Once the Subsystem Permission is checked against the caller the Subsystems implementation will proceed with the actual Subsystem operation. This operation will require a number of other permissions to complete. For example, the Admin Permission will be needed to install, start, stop, and uninstall resources of type osgi.bundle for a Subsystem. The Subsystems implementation must isolate the caller from such permission checks by use of a proper `doPrivileged` block.

### 134.20 org.osgi.service.subsystem

Subsystem Service Package Version 1.1.

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.subsystem; version="[1.1,2.0)"
```

**Example import for providers implementing the API in this package:**

```
Import-Package: org.osgi.service.subsystem; version="[1.1,1.2)"
```

### 134.20.1 Summary

- **Subsystem** - A subsystem is a collection of resources constituting a logical, possibly isolated, unit of functionality.
- **Subsystem.State** - An enumeration of the possible states of a subsystem.
- **SubsystemConstants** - Defines the constants used by Subsystem service property, manifest header, attribute and directive keys.
- **SubsystemException** - A Subsystem exception used to indicate a problem.
- **SubsystemPermission** - A bundle's authority to perform specific privileged administrative operations on or to get sensitive information about a subsystem.
A subsystem is a collection of resources constituting a logical, possibly isolated, unit of functionality.

A subsystem may be scoped or unscoped. Scoped subsystems are isolated by implicit or explicit sharing policies. Unscoped subsystems are not isolated and, therefore, have no sharing policy. There are three standard types of subsystems.

- **Application** - An implicitly scoped subsystem. Nothing is exported, and imports are computed based on any unsatisfied content requirements.
- **Composite** - An explicitly scoped subsystem. The sharing policy is defined by metadata within the subsystem archive.
- **Feature** - An unscoped subsystem.

Conceptually, a subsystem may be thought of as existing in an isolated region along with zero or more other subsystems. Each region has one and only one scoped subsystem, which dictates the sharing policy. The region may, however, have many unscoped subsystems. It is, therefore, possible to have shared constituents across multiple subsystems within a region. Associated with each region is a bundle whose context may be retrieved from any subsystem within that region. This context may be used to monitor activity occurring within the region.

A subsystem may have children and, unless it’s the root subsystem, must have at least one parent. Subsystems become children of the subsystem in which they are installed. Unscoped subsystems have more than one parent if they are installed in more than one subsystem within the same region. The subsystem graph may be thought of as an acyclic digraph [http://en.wikipedia.org/wiki/Directed_acyclic_graph] with one and only one source vertex, which is the root subsystem. The edges have the child as the head and parent as the tail.

A subsystem has several identifiers.

- **Location** - An identifier specified by the client as part of installation. It is guaranteed to be unique within the same framework.
- **ID** - An identifier generated by the implementation as part of installation. It is guaranteed to be unique within the same framework.
- **Symbolic Name/Version** - The combination of symbolic name and version is guaranteed to be unique within the same region. Although type is not formally part of the identity, two subsystems with the same symbolic names and versions but different types are not considered to be equal.

A subsystem has a well-defined life cycle. Which stage a subsystem is in may be obtained from the subsystem’s state and is dependent on which life cycle operation is currently active or was last invoked.

A subsystem archive is a ZIP file having an .esa extension and containing metadata describing the subsystem. The form of the metadata may be a subsystem or deployment manifest, as well as any content resource files. The manifests are optional and will be computed if not present. The subsystem manifest headers may be retrieved in raw or localized forms. There are five standard types of resources that may be included in a subsystem.

- **Bundle** - A bundle that is not a fragment.
- **Fragment** - A fragment bundle.
- **Application Subsystem** - An application subsystem.
- **Composite Subsystem** - A composite subsystem.
- **Feature Subsystem** - A feature subsystem.

Resources contained by a subsystem are called constituents. There are several ways a resource may become a constituent of a subsystem:
A resource is listed as part of the subsystem's content.
A subsystem resource is a child of the subsystem.
The subsystem has a provision policy of accept dependencies.
A bundle resource is installed using the region bundle context.
A bundle resource is installed using the bundle context of another resource contained by the subsystem.

In addition to invoking one of the install methods, a subsystem instance may be obtained through the service registry. Each installed subsystem has a corresponding service registration. A subsystem service has the following properties.

- **ID** - The ID of the subsystem.
- **Symbolic Name** - The symbolic name of the subsystem.
- **Version** - The version of the subsystem.
- **Type** - The type of the subsystem.
- **State** - The state of the subsystem.

Because a subsystem must be used to install other subsystems, a root subsystem is provided as a starting point. The root subsystem may only be obtained as a service and has the following characteristics.

- The ID is 0.
- The symbolic name is org.osgi.service.subsystem.root.
- The version matches this specification's version.
- It has no parents.
- All existing bundles, including the system and subsystem implementation bundles, are constituents.
- The type is osgi.subsystem.application with no imports.
- The provision policy is acceptDependencies.

### Concurrency
Thread-safe

### Provider Type
Consumers of this API must not implement this type

#### 134.20.2.1 public BundleContext getBundleContext()

- Returns the bundle context of the region within which this subsystem resides.

  The bundle context offers the same perspective of any resource contained by a subsystem within the region. It may be used, for example, to monitor events internal to the region as well as external events visible to the region. All subsystems within the same region have the same bundle context. If this subsystem is in a state where the bundle context would be invalid, null is returned.

  **Returns** The bundle context of the region within which this subsystem resides or null if this subsystem's state is in INSTALL_FAILED, UNINSTALLED.

  **Throws** SecurityException – If the caller does not have the appropriate SubsystemPermission[this,CONTEXT], and the runtime supports permissions.

#### 134.20.2.2 public Collection<Subsystem> getChildren()

- Returns the child subsystems of this subsystem.

  The child subsystems of this subsystem. The returned collection is an unmodifiable snapshot of all subsystems that are installed in this subsystem. The collection will be empty if no subsystems are installed in this subsystem.

  **Returns** The child subsystems of this subsystem. The returned collection is an unmodifiable snapshot of all subsystems that are installed in this subsystem. The collection will be empty if no subsystems are installed in this subsystem.

  **Throws** IllegalStateException – If this subsystem's state is in INSTALL_FAILED, UNINSTALLED.
134.20.2.3  public Collection<Resource> getConstituents()

 □ Returns the constituent resources of this subsystem.

 Returns The constituent resources of this subsystem. The returned collection is an unmodifiable snapshot of the constituent resources of this subsystem. If this subsystem has no constituents, the collection will be empty.

 Throws IllegalStateException – If this subsystem's state is in INSTALL_FAILED, UNINSTALLED.

134.20.2.4  public Map<String,String> getDeploymentHeaders()

 □ Returns the headers for this subsystem's deployment manifest.

 Each key in the map is a header name and the value of the key is the corresponding header value. Because header names are case-insensitive, the methods of the map must treat the keys in a case-insensitive manner. If the header name is not found, null is returned. Both original and derived headers will be included in the map.

 This method must continue to return the headers while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

 Returns The headers for this subsystem's deployment manifest. The returned map is unmodifiable.

 Throws SecurityException – If the caller does not have the appropriate SubsystemPermission[this,METADATA], and the runtime supports permissions.

 Since 1.1

134.20.2.5  public String getLocation()

 □ Returns the location identifier of this subsystem.

 The location identifier is the location that was passed to the install method of the parent subsystem. It is unique within the framework.

 This method must continue to return this subsystem's headers while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

 Returns The location identifier of this subsystem.

 Throws SecurityException – If the caller does not have the appropriate SubsystemPermission[this,METADATA], and the runtime supports permissions.

134.20.2.6  public Collection<Subsystem> getParents()

 □ Returns the parent subsystems of this subsystem.

 Returns The parent subsystems of this subsystem. The returned collection is an unmodifiable snapshot of all subsystems in which this subsystem is installed. The collection will be empty for the root subsystem; otherwise, it must contain at least one parent. Scoped subsystems always have only one parent. Unscoped subsystems may have multiple parents.

 Throws IllegalStateException – If this subsystem's state is in INSTALL_FAILED, UNINSTALLED.

134.20.2.7  public Subsystem.State getState()

 □ Returns the current state of this subsystem.

 Returns The current state of this subsystem.
134.20.2.8 public Map<String, String> getSubsystemHeaders(Locale locale)

locale The locale for which translations are desired. The header values are translated according to the specified locale. If the specified locale is null or not supported, the raw values are returned. If the translation for a particular header is not found, the raw value is returned.

Returns The headers for this subsystem's subsystem manifest. Each key in the map is a header name and the value of the key is the corresponding header value. Because header names are case-insensitive, the methods of the map must treat the keys in a case-insensitive manner. If the header name is not found, null is returned. Both original and derived headers will be included in the map.

This method must continue to return the headers while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

Throws SecurityException – If the caller does not have the appropriate SubsystemPermission[this,METADATA], and the runtime supports permissions.

134.20.2.9 public long getSubsystemId()

Returns the identifier of this subsystem. The identifier is a monotonically increasing, non-negative integer automatically generated at installation time and guaranteed to be unique within the framework. The identifier of the root subsystem is zero.

This method must continue to return this subsystem's identifier while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

Returns The identifier of this subsystem.

134.20.2.10 public String getSymbolicName()

Returns the symbolic name of this subsystem. The subsystem symbolic name conforms to the same grammar rules as the bundle symbolic name and is derived from one of the following, in order.

- The value of the Subsystem-SymbolicName header, if specified.
- The subsystem URI if passed as the location along with the content to the install method.
- Optionally generated in an implementation specific way.

The combination of subsystem symbolic name and version is unique within a region. The symbolic name of the root subsystem is org.osgi.service.subsystem.root.

This method must continue to return this subsystem's symbolic name while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

Returns The symbolic name of this subsystem.

134.20.2.11 public String getType()

Returns the type of this subsystem.

This method must continue to return this subsystem's type while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

Returns The type of this subsystem.

134.20.2.12 public Version getVersion()

Returns the version of this subsystem.
The subsystem version conforms to the same grammar rules as the bundle version and is derived from one of the following, in order.

- The value of the Subsystem-Version header, if specified.
- The subsystem URI if passed as the location along with the content to the install method.
- Defaults to 0.0.0.

The combination of subsystem symbolic name and version is unique within a region. The version of the root subsystem matches this specification's version.

This method must continue to return this subsystem's version while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

**Returns**
The version of this subsystem.

**134.20.2.13**

```java
public Subsystem install(String location)
```

**location**
The location identifier of the subsystem to install.

- Installs a subsystem from the specified location identifier.
- This method performs the same function as calling `install(String, InputStream)` with the specified location identifier and null as the content.

**Returns**
The installed subsystem.

**Throws**
- `IllegalStateException` – If this subsystem's state is in INSTALLING, INSTALL_FAILED, UNINSTALLING, UNINSTALLED.
- `SubsystemException` – If the installation failed.
- `SecurityException` – If the caller does not have the appropriate SubsystemPermission for installed subsystem.LIFECYCLE, and the runtime supports permissions.

**See Also**
`install(String, InputStream)`

**134.20.2.14**

```java
public Subsystem install(String location, InputStream content)
```

- **location**
The location identifier of the subsystem to be installed.
- **content**
The input stream from which this subsystem will be read or null to indicate the input stream must be created from the specified location identifier. The input stream will always be closed when this method completes, even if an exception is thrown.

- Installs a subsystem from the specified content.

  The specified location will be used as an identifier of the subsystem. Every installed subsystem is uniquely identified by its location, which is typically in the form of a URI. If the specified location conforms to the `subsystem-uri` grammar, the required symbolic name and optional version information will be used as default values.

  If the specified content is null, a new input stream must be created from which to read the subsystem by interpreting, in an implementation dependent manner, the specified location.

  A subsystem installation must be persistent. That is, an installed subsystem must remain installed across Framework and VM restarts.

  All references to changing the state of this subsystem include both changing the state of the subsystem object as well as the state property of the subsystem service registration.

  The following steps are required to install a subsystem.

  1. If an installed subsystem with the specified location identifier already exists, return the installed subsystem.

  2. Read the specified content in order to determine the symbolic name, version, and type of the installing subsystem. If an error occurs while reading the content, an installation failure results.
3. If an installed subsystem with the same symbolic name and version already exists within this subsystem's region, complete the installation with one of the following.
   • If the installing and installed subsystems' types are not equal, an installation failure results.
   • If the installing and installed subsystems' types are equal, and the installed subsystem is already a child of this subsystem, return the installed subsystem.
   • If the installing and installed subsystems' types are equal, and the installed subsystem is not already a child of this subsystem, add the installed subsystem as a child of this subsystem, increment the installed subsystem's reference count by one, and return the installed subsystem.

4. Create a new subsystem based on the specified location and content.
5. If the subsystem is scoped, install and start a new region context bundle.
6. Change the state to INSTALLING and register a new subsystem service.
7. Discover the subsystem's content resources. If any mandatory resource is missing, an installation failure results.
8. Discover the dependencies required by the content resources. If any mandatory dependency is missing, an installation failure results.
9. Using a framework ResolverHook, disable runtime resolution for the resources.
10. For each resource, increment the reference count by one. If the reference count is one, install the resource. If an error occurs while installing a resource, an install failure results with that error as the cause.
11. If the subsystem is scoped, enable the import sharing policy.
12. Enable runtime resolution for the resources.
13. Change the state of the subsystem to INSTALLED.
14. Return the new subsystem.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption. An interrupted thread should result in a SubsystemException with an InterruptedException as the cause and be treated as an installation failure.

All installation failure flows include the following, in order.
1. Change the state to INSTALL_FAILED.
2. Change the state to UNINSTALLING.
3. All content and dependencies which may have been installed by the installing process must be uninstalled.
4. Change the state to UNINSTALLED.
5. Unregister the subsystem service.
6. If the subsystem is a scoped subsystem then, uninstall the region context bundle.
7. Throw a SubsystemException with the cause of the installation failure.

Returns
The installed subsystem.

Throws
IllegalArgumentException – If this subsystem's state is in INSTALLING, INSTALL_FAILED, UNINSTALLING, UNINSTALLED.
SubsystemException – If the installation failed.
SecurityException – If the caller does not have the appropriate SubsystemPermission[installed subsystem,LIFECYCLE], and the runtime supports permissions.

134.20.2.15

public Subsystem install(String location,InputStream content,InputStream deploymentManifest)

location The location identifier of the subsystem to be installed.
content The input stream from which this subsystem will be read or null to indicate the input stream must be created from the specified location identifier. The input stream will always be closed when this method completes, even if an exception is thrown.
**deploymentManifest**

The deployment manifest to use in lieu of the one in the archive, if any, or a computed one.

- Installs a subsystem from the specified content according to the specified deployment manifest.

This method installs a subsystem using the provided deployment manifest instead of the one in the archive, if any, or a computed one. If the deployment manifest is null, the behavior is exactly the same as in the `install(String, InputStream)` method. Implementations must support deployment manifest input streams in the format described by section 134.2 of the Subsystem Service Specification. If the deployment manifest does not conform to the subsystem manifest (see 134.15.2), the installation fails.

**Returns**

The installed subsystem.

**Throws**

- `IllegalStateException` – If this subsystem's state is in INSTALLING, INSTALL FAILED, UNINSTALLING, UNINSTALLED.
- `SubsystemException` – If the installation failed.
- `SecurityException` – If the caller does not have the appropriate SubsystemPermission[installed subsystem, LIFECYCLE], and the runtime supports permissions.

Since 1.1

134.20.2.16

**public void start()**

- Starts this subsystem.

The following table shows which actions are associated with each state. An action of Wait means this method will block until a state transition occurs, upon which the new state will be evaluated in order to determine how to proceed. If a state transition does not occur in a reasonable time while waiting then no action is taken and a SubsystemException is thrown to indicate the subsystem was unable to be started. An action of Return means this method returns immediately without taking any other action.

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALLING</td>
<td>Wait</td>
</tr>
<tr>
<td>INSTALLED</td>
<td>Resolve, Start</td>
</tr>
<tr>
<td>INSTALL FAILED</td>
<td>IllegalStateException</td>
</tr>
<tr>
<td>RESOLVING</td>
<td>Wait</td>
</tr>
<tr>
<td>RESOLVED</td>
<td>Start</td>
</tr>
<tr>
<td>STARTING</td>
<td>Wait</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>Return</td>
</tr>
<tr>
<td>STOPPING</td>
<td>Wait</td>
</tr>
<tr>
<td>UNINSTALLING</td>
<td>IllegalStateException</td>
</tr>
<tr>
<td>UNINSTALLED</td>
<td>IllegalStateException</td>
</tr>
</tbody>
</table>

All references to changing the state of this subsystem include both changing the state of the subsystem object as well as the state property of the subsystem service registration.

A subsystem must be persistently started. That is, a started subsystem must be restarted across Framework and VM restarts, even if a start failure occurs.

The following steps are required to start this subsystem.

1. Set the subsystem autostart setting to started.
2. If this subsystem is in the RESOLVED state, proceed to step 7.
3. Change the state to RESOLVING.
4. Resolve the content resources. A resolution failure results in a start failure with a state of INSTALLED.
5. Change the state to RESOLVED.
6. If this subsystem is scoped, enable the export sharing policy.
7. Change the state to STARTING.
8. For each eligible resource, increment the active use count by one. If the active use count is one, start the resource. All dependencies must be started before any content resource, and content resources must be started according to the specified start order. If an error occurs while starting a resource, a start failure results with that error as the cause.
9. Change the state to ACTIVE.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption. An interrupted thread should be treated as a start failure with an InterruptedException as the cause.

All start failure flows include the following, in order.
1. If the subsystem state is STARTING then change the state to STOPPING and stop all resources that were started as part of this operation.
2. Change the state to either INSTALLED or RESOLVED.
3. Throw a SubsystemException with the specified cause.

Throws  SubsystemException—If this subsystem fails to start.

IllegalStateException—If this subsystem’s state is in INSTALL_FAILED, UNINSTALLING, or UNINSTALLED, or if the state of at least one of this subsystem’s parents is not in STARTING, ACTIVE.

SecurityException—If the caller does not have the appropriate SubsystemPermission[this,EXECUTE], and the runtime supports permissions.

134.20.2.17 public void stop()

□ Stops this subsystem.

The following table shows which actions are associated with each state. An action of Wait means this method will block until a state transition occurs, upon which the new state will be evaluated in order to determine how to proceed. If a state transition does not occur in a reasonable time while waiting then no action is taken and a SubsystemException is thrown to indicate the subsystem was unable to be stopped. An action of Return means this method returns immediately without taking any other action.

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALLING</td>
<td>Wait</td>
</tr>
<tr>
<td>INSTALLED</td>
<td>Return</td>
</tr>
<tr>
<td>INSTALL_FAILED</td>
<td>IllegalStateException</td>
</tr>
<tr>
<td>RESOLVING</td>
<td>Wait</td>
</tr>
<tr>
<td>RESOLVED</td>
<td>Return</td>
</tr>
<tr>
<td>STARTING</td>
<td>Wait</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>Stop</td>
</tr>
<tr>
<td>STOPPING</td>
<td>Wait</td>
</tr>
<tr>
<td>UNINSTALLING</td>
<td>IllegalStateException</td>
</tr>
<tr>
<td>UNINSTALLED</td>
<td>IllegalStateException</td>
</tr>
</tbody>
</table>

A subsystem must be persistently stopped. That is, a stopped subsystem must remain stopped across Framework and VM restarts.

All references to changing the state of this subsystem include both changing the state of the subsystem object as well as the state property of the subsystem service registration.
The following steps are required to stop this subsystem.

1. Set the subsystem `autostart setting` to `stopped`.
2. Change the state to STOPPING.
3. For each eligible resource, decrement the active use count by one. If the active use count is zero, stop the resource. All content resources must be stopped before any dependencies, and content resources must be stopped in reverse start order.
4. Change the state to RESOLVED.

With regard to error handling, once this subsystem has transitioned to the STOPPING state, every part of each step above must be attempted. Errors subsequent to the first should be logged. Once the stop process has completed, a SubsystemException must be thrown with the initial error as the specified cause.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption, in which case a SubsystemException with an InterruptedException as the cause should be thrown. If an interruption occurs while waiting, this method should terminate immediately. Once the transition to the STOPPING state has occurred, however, this method must not terminate due to an interruption until the stop process has completed.

`Throws` SubsystemException—If this subsystem fails to stop cleanly.

IllegalStateException—If this subsystem’s state is in INSTALL_FAILED, UNINSTALLING, or UNINSTALLED.

SecurityException—If the caller does not have the appropriate SubsystemPermission[this,EXECUTE], and the runtime supports permissions.

134.20.2.18 public void uninstall()

☐ Uninstalls this subsystem.

The following table shows which actions are associated with each state. An action of `Wait` means this method will block until a state transition occurs, upon which the new state will be evaluated in order to determine how to proceed. If a state transition does not occur in a reasonable time while waiting then no action is taken and a SubsystemException is thrown to indicate the subsystem was unable to be uninstalled. An action of `Return` means this method returns immediately without taking any other action.

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALLING</td>
<td>Wait</td>
</tr>
<tr>
<td>INSTALLED</td>
<td>Uninstall</td>
</tr>
<tr>
<td>INSTALL_FAILED</td>
<td>Wait</td>
</tr>
<tr>
<td>RESOLVING</td>
<td>Wait</td>
</tr>
<tr>
<td>RESOLVED</td>
<td>Uninstall</td>
</tr>
<tr>
<td>STARTING</td>
<td>Wait</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>Stop, Uninstall</td>
</tr>
<tr>
<td>STOPPING</td>
<td>Wait</td>
</tr>
<tr>
<td>UNINSTALLING</td>
<td>Wait</td>
</tr>
<tr>
<td>UNINSTALLED</td>
<td>Return</td>
</tr>
</tbody>
</table>

All references to changing the state of this subsystem include both changing the state of the subsystem object as well as the state property of the subsystem service registration.

The following steps are required to uninstall this subsystem after being stopped if necessary.

1. Change the state to INSTALLED.
2. Change the state to UNINSTALLING.
3. For each referenced resource, decrement the reference count by one. If the reference count is zero, uninstall the resource. All content resources must be uninstalled before any dependencies.
4. Change the state to UNINSTALLED.
5. Unregister the subsystem service.
6. If the subsystem is scoped, uninstall the region context bundle.

With regard to error handling, once this subsystem has transitioned to the UNINSTALLING state, every part of each step above must be attempted. Errors subsequent to the first should be logged. Once the uninstall process has completed, a SubsystemException must be thrown with the specified cause.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption, in which case a SubsystemException with an InterruptedException as the cause should be thrown. If an interruption occurs while waiting, this method should terminate immediately. Once the transition to the UNINSTALLING state has occurred, however, this method must not terminate due to an interruption until the uninstall process has completed.

**Throws**

SubsystemException – If this subsystem fails to uninstall cleanly.
SecurityException – If the caller does not have the appropriate SubsystemPermission[this,LIFECYCLE], and the runtime supports permissions.

### 134.20.3 enum Subsystem.State

An enumeration of the possible states of a subsystem.

These states are a reflection of what constituent resources are permitted to do and not an aggregation of constituent resource states.

#### 134.20.3.1 INSTALLING

The subsystem is in the process of installing.

A subsystem is in the INSTALLING state when the install method of its parent is active, and attempts are being made to install its content resources. If the install method completes without exception, then the subsystem has successfully installed and must move to the INSTALLED state. Otherwise, the subsystem has failed to install and must move to the INSTALL_FAILED state.

#### 134.20.3.2 INSTALLED

The subsystem is installed but not yet resolved.

A subsystem is in the INSTALLED state when it has been installed in a parent subsystem but is not or cannot be resolved. This state is visible if the dependencies of the subsystem's content resources cannot be resolved.

#### 134.20.3.3 INSTALL_FAILED

The subsystem failed to install.

A subsystem is in the INSTALL_FAILED state when an unrecoverable error occurred during installation. The subsystem is in an unusable state but references to the subsystem object may still be available and used for introspection.

#### 134.20.3.4 RESOLVING

The subsystem is in the process of resolving.

A subsystem is in the RESOLVING state when attempts are being made to resolve its content resources. If the resolve process completes without exception, then the subsystem has successfully resolved and must move to the RESOLVED state. Otherwise, the subsystem has failed to resolve and must move to the INSTALLED state.
The subsystem is resolved and able to be started.
A subsystem is in the RESOLVED state when all of its content resources are resolved. Note that the subsystem is not active yet.

The subsystem is in the process of starting.
A subsystem is in the STARTING state when its start method is active, and attempts are being made to start its content and dependencies. If the start method completes without exception, then the subsystem has successfully started and must move to the ACTIVE state. Otherwise, the subsystem has failed to start and must move to the RESOLVED state.

The subsystem is now running.
A subsystem is in the ACTIVE state when its content and dependencies have been successfully started.

The subsystem is in the process of stopping.
A subsystem is in the STOPPING state when its stop method is active, and attempts are being made to stop its content and dependencies. When the stop method completes, the subsystem is stopped and must move to the RESOLVED state.

The subsystem is in the process of uninstalling.
A subsystem is in the UNINSTALLING state when its uninstall method is active, and attempts are being made to uninstall its constituent and dependencies. When the uninstall method completes, the subsystem is uninstalled and must move to the UNINSTALLED state.

The subsystem is uninstalled and may not be used.
The UNINSTALLED state is only visible after a subsystem’s constituent and dependencies are uninstalled. The subsystem is in an unusable state but references to the subsystem object may still be available and used for introspection.

public class SubsystemConstants
Defines the constants used by Subsystem service property, manifest header, attribute and directive keys.
The values associated with these keys are of type String, unless otherwise indicated.

Concurrency Immutable

public static final String DEPLOYED_CONTENT = "Deployed-Content"
Manifest header identifying the resources to be deployed.

public static final String DEPLOYED_VERSION_ATTRIBUTE = "deployed-version"
Manifest header attribute identifying the deployed version.

public static final String DEPLOYMENT_MANIFESTVERSION = "Deployment-ManifestVersion"
Manifest header identifying the deployment manifest version. If not present, the default value is 1.
public static final String PREFERRED_PROVIDER = "Preferred-Provider"
Manifest header used to express a preference for particular resources to satisfy implicit package de-
pendencies.

public static final String PROVISION_POLICY_ACCEPT_DEPENDENCIES = "acceptDependencies"
A value for the provision-policy directive indicating the subsystem accepts dependency resources. The root subsystem has this provision policy.

public static final String PROVISION_POLICY_DIRECTIVE = "provision-policy"
Manifest header directive identifying the provision policy. The default value is rejectDependencies
See Also PROVISION_POLICY_ACCEPT_DEPENDENCIES, PROVISION_POLICY_REJECT_DEPENDENCIES

public static final String PROVISION_POLICY_REJECT_DEPENDENCIES = "rejectDependencies"
A value for the provision-policy directive indicating the subsystem does not accept dependency re-
sources. This is the default value.

public static final String PROVISION_RESOURCE = "Provision-Resource"
Manifest header identifying the resources to be deployed to satisfy the dependencies of a subsystem.

public static final String ROOT_SUBSYSTEM_SYMBOLICNAME = "org.osgi.service.subsystem.root"
The symbolic name of the root subsystem.

public static final String START_ORDER_DIRECTIVE = "start-order"
Manifest header directive identifying the start order of subsystem contents. There is no default val-
ue. Specified values are of type String and must represent an integer.

public static final String SUBSYSTEM_CATEGORY = "Subsystem-Category"
Manifest header identifying the categories of a subsystem as a comma-delimited list.
Since 1.1

public static final String SUBSYSTEM_CONTACTADDRESS = "Subsystem-ContactAddress"
Manifest header identifying the contact address where problems with a subsystem may be reported; for example, an email address.
Since 1.1

public static final String SUBSYSTEM_CONTENT = "Subsystem-Content"
Manifest header identifying the list of subsystem contents identified by a symbolic name and ver-

public static final String SUBSYSTEM_COPYRIGHT = "Subsystem-Copyright"
Manifest header identifying a subsystem's copyright information.
Since 1.1

public static final String SUBSYSTEM_DESCRIPTION = "Subsystem-Description"
Manifest header identifying the human readable description.

public static final String SUBSYSTEM_DOCURL = "Subsystem-DocURL"
Manifest header identifying a subsystem's documentation URL, from which further information about the subsystem may be obtained.
Since 1.1
134.20.4.17  public static final String SUBSYSTEM_EXPORTSERVICE = "Subsystem-ExportService"
Manifest header identifying services offered for export.

134.20.4.18  public static final String SUBSYSTEM_ICON = "Subsystem-Icon"
Manifest header identifying the icon URL for the subsystem.
Since 1.1

134.20.4.19  public static final String SUBSYSTEM_ID_PROPERTY = "subsystem.id"
The name of the service property for the subsystem ID. The value of this property must be of type Long.

134.20.4.20  public static final String SUBSYSTEM_IMPORTSERVICE = "Subsystem-ImportService"
Manifest header identifying services required for import.

134.20.4.21  public static final String SUBSYSTEM_LICENSE = "Subsystem-License"
Manifest header identifying a subsystem's license.
Since 1.1

134.20.4.22  public static final String SUBSYSTEM_LOCALIZATION = "Subsystem-Localization"
Manifest header identifying the base name of a subsystem's localization entries.
Since 1.1

134.20.4.23  public static final String SUBSYSTEM_LOCALIZATION_DEFAULT_BASENAME = "OSGI-INF/l10n/subsystem"
Default value for the Subsystem-Localization manifest header.
Since 1.1

134.20.4.24  public static final String SUBSYSTEM_MANIFESTVERSION = "Subsystem-ManifestVersion"
Manifest header identifying the subsystem manifest version. If not present, the default value is 1.

134.20.4.25  public static final String SUBSYSTEM_NAME = "Subsystem-Name"
Manifest header identifying the human readable subsystem name.

134.20.4.26  public static final String SUBSYSTEM_STATE_PROPERTY = "subsystem.state"
The name of the service property for the subsystem state. The value of this property must be of type Subsystem.State.

134.20.4.27  public static final String SUBSYSTEM_SYMBOLICNAME = "Subsystem-SymbolicName"
Manifest header value identifying the symbolic name for the subsystem. Must be present.

134.20.4.28  public static final String SUBSYSTEM_SYMBOLICNAME_PROPERTY = "subsystem.symbolicName"
The name of the service property for the subsystem symbolic name.

134.20.4.29  public static final String SUBSYSTEM_TYPE = "Subsystem-Type"
Manifest header identifying the subsystem type.
See Also SUBSYSTEM_TYPE_APPLICATION, SUBSYSTEM_TYPE_COMPOSITE, SUBSYSTEM_TYPE_FEATURE

134.20.4.30  public static final String SUBSYSTEM_TYPE_APPLICATION = "osgi.subsystem.application"
The resource type value identifying an application subsystem.
This value is used for the osgi.identity capability attribute type, the SUBSYSTEM_TYPE manifest header and the SUBSYSTEM_TYPE_PROPERTY service property.

### 134.20.431 public static final String SUBSYSTEM_TYPE_COMPOSITE = "osgi.subsystem.composite"
The resource type value identifying a composite subsystem.

### 134.20.432 public static final String SUBSYSTEM_TYPE_FEATURE = "osgi.subsystem.feature"
The resource type value identifying a feature subsystem.

### 134.20.433 public static final String SUBSYSTEM_TYPE_PROPERTY = "subsystem.type"
The name of the service property for the subsystem type.

### 134.20.434 public static final String SUBSYSTEM_VENDOR = "Subsystem-Vendor"
Manifest header identifying a subsystem's vendor.

#### Since 1.1

### 134.20.435 public static final String SUBSYSTEM_VERSION = "Subsystem-Version"
Manifest header value identifying the version of the subsystem. If not present, the default value is 0.0.0.

### 134.20.436 public static final String SUBSYSTEM_VERSION_PROPERTY = "subsystem.version"
The name of the service property for the subsystem version. The value of this property must be of type Version.

### 134.20.5 public class SubsystemException extends RuntimeException
A Subsystem exception used to indicate a problem.

### 134.20.5.1 public SubsystemException()
Construct a Subsystem exception with no message.

### 134.20.5.2 public SubsystemException(String message)

#### message
The message to include in the exception.

### 134.20.5.3 public SubsystemException(Throwable cause)

#### cause
The cause of the exception.

### 134.20.5.4 public SubsystemException(String message, Throwable cause)

#### message
The message to include in the exception.

#### cause
The cause of the exception.
Construct a Subsystem exception specifying a message and a cause.

134.20.6

**public final class SubsystemPermission**

**extends BasicPermission**

A bundle's authority to perform specific privileged administrative operations on or to get sensitive information about a subsystem. The actions for this permission are:

<table>
<thead>
<tr>
<th>Action</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>Subsystem.getBundleContext</td>
</tr>
<tr>
<td>execute</td>
<td>Subsystem.start</td>
</tr>
<tr>
<td></td>
<td>Subsystem.stop</td>
</tr>
<tr>
<td>lifecycle</td>
<td>Subsystem.install</td>
</tr>
<tr>
<td></td>
<td>Subsystem.uninstall</td>
</tr>
<tr>
<td>metadata</td>
<td>Subsystem.getSubsystemHeaders</td>
</tr>
<tr>
<td></td>
<td>Subsystem.getLocation</td>
</tr>
</tbody>
</table>

The name of this permission is a filter expression. The filter gives access to the following attributes:

- location - The location of a subsystem.
- id - The subsystem ID of the designated subsystem.
- name - The symbolic name of a subsystem.

Filter attribute names are processed in a case sensitive manner.

**Concurrency**  
Thread-safe

134.20.6.1 **public static final String CONTEXT** = "context"

The action string context.

134.20.6.2 **public static final String EXECUTE** = "execute"

The action string execute.

134.20.6.3 **public static final String LIFECYCLE** = "lifecycle"

The action string lifecycle.

134.20.6.4 **public static final String METADATA** = "metadata"

The action string metadata.

134.20.6.5 **public SubsystemPermission(String filter, String actions)**

**filter** A filter expression that can use, location, id, and name keys. Filter attribute names are processed in a case sensitive manner. A special value of "*" can be used to match all subsystems.

**actions** execute, lifecycle, metadata, or context.

- Create a new SubsystemPermission. This constructor must only be used to create a permission that is going to be checked.

Examples:

(name=com.acme.*) (location=http://www.acme.com/subsystems/*))

(id>=1)

**Throws**  
IllegalArgumentException – If the filter has an invalid syntax.

134.20.6.6 **public SubsystemPermission(Subsystem subsystem, String actions)**

**subsystem** A subsystem.
actions execute, lifecycle, metadata, or context.

- Creates a new requested SubsystemPermission object to be used by the code that must perform checkPermission. SubsystemPermission objects created with this constructor cannot be added to an SubsystemPermission permission collection.

134.20.6.7 public boolean equals(Object obj)

- The object being compared for equality with this object.
- Determines the equality of two SubsystemPermission objects.

Returns true if obj is equivalent to this SubsystemPermission; false otherwise.

134.20.6.8 public String getActions()

- Returns the canonical string representation of the SubsystemPermission actions.
- Always returns present SubsystemPermission actions in the following order: execute, lifecycle, metadata, context.

Returns Canonical string representation of the SubsystemPermission actions.

134.20.6.9 public int hashCode()

- Returns the hash code value for this object.

Returns Hash code value for this object.

134.20.6.10 public boolean implies(Permission p)

- The requested permission.
- Determines if the specified permission is implied by this object. This method throws an exception if the specified permission was not constructed with a subsystem.

This method returns true if the specified permission is a SubsystemPermission AND

- this object's filter matches the specified permission's subsystem ID, subsystem symbolic name, and subsystem location OR
- this object's filter is "*"

AND this object's actions include all of the specified permission's actions.

Special case: if the specified permission was constructed with "*" filter, then this method returns true if this object's filter is "*" and this object's actions include all of the specified permission's actions

Returns true if the specified permission is implied by this object; false otherwise.

134.20.6.11 public PermissionCollection newPermissionCollection()

- Returns a new PermissionCollection object suitable for storing SubsystemPermissions.

Returns A new PermissionCollection object.

134.21 References

[1] Filter Based Permissions
OSGi Core, Chapter 2, Filter Based Permissions

[2] Core Service Hooks
Changes

- Specified the translation of subsystem manifest headers.
- Added more informational subsystem manifest headers.
- Specified the interaction between Subsystems and Weaving Hooks for the purposes of supporting dynamically added import packages.
- Added the ability to override the archive’s deployment manifest at install time.
- Added the ability to retrieve deployment manifest headers.
- Specified a means for declaratively identifying the services a bundle provides.
- Modified the procedure for determining dependencies to include the capabilities, in a specified order, from all repositories in order to increase the likelihood that a resolution will be found such as when a uses constraint violation would otherwise result.
135 Common Namespaces Specification

Version 1.0

135.1 Introduction

A key aspect of the OSGi general dependency model based on requirements and capabilities is the concept of a Namespace. A Namespace defines the semantics of a Requirement-Capability pair. The generic model is defined in the [2] Resources API Specification. This section defines a number of Namespaces that are not part of the OSGi Core Release 6 specification. Unless an attribute is specifically overridden, all Namespaces inherit the attributes and directives of the default Namespace as defined [3] Framework Namespaces Specification.

Each Namespace is defined with the following items:

- **Name** - the name of an attribute or directive
- **Kind** - Defines where the attribute or directive can be used
  - CA - Capability Attribute
  - CD - Capability Directive
  - RA - Requirement Attribute
  - RD - Requirement Directive
- **M/O** - Mandatory (M) or Optional (O)
- **Type** - The data type

135.2 osgi.extender Namespace

An Extender is a bundle that uses the life cycle events from another bundle, the extendee, to extend that bundle’s functionality when that bundle is active. It can use metadata (headers, or files inside the extendee) to control its functionality. Extendees therefore have a dependency on the Extender that can be modeled with the osgi.extender namespace. The definition for this namespace can be found in the following table and the ExtenderNamespace class.

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>M/O</th>
<th>Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>osgi.extender</td>
<td>CA</td>
<td>M</td>
<td>String</td>
<td>symbolic-name</td>
<td>A symbolic name for the extender. These names are defined in their respective specifications and should in general use the specification top level package name. For example, org.acme.foo. The OSGi Alliance reserves names that start with osgi..</td>
</tr>
</tbody>
</table>
### osgi.contract Namespace

Products or technologies often have a number of related APIs consisting of a large set of packages. Some IDEs have not optimized for OSGi and requires work for each imported package. In these development environments using modularized systems tends to require a significant amount of manual effort to manage the imported packages.

The osgi.contract Namespace addresses this IDE deficiency. It allows a developer to specify a single name and version for a contract that can then be expanded to a potentially large number of packages. For example, a developer can then specify a dependency on Java Enterprise Edition 6 contract that can be provided by an application server.

The osgi.contract Namespace provides such a name and binds it to a set of packages with the uses constraint. The bundle that declares this contract must then import or export each of the listed packages with the correct versioning. Such a bundle is called a **contract bundle**. The contract bundle must ensure that it is bound to the correct versions of the packages contained within the contract it is providing. If the contract bundle imports the packages which are specified as part of the contract then proper matching attributes must be used to make sure it is bound to the correct versions of the packages.

An osgi.contract capability can then be used in the following ways:

- IDEs can use the information in the uses directive to make all those packages available on the build path. In this case the developer no longer has to specify each package separately.
- During run time the uses clause is used to enforce that all packages in the contract form a consistent class space.

The uses directive will make it impossible to get wired to packages that are not valid for the contract. Since the uses constrains enforce the consistency, it is in principle not necessary to version the imported packages on client bundles since only the correctly versioned packages can be used. Contracts are aggregates and therefore make clients depend on the whole and all their transitive dependencies, even if the client only uses a single package of the contract.

### Specifications

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>O/M</th>
<th>Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>CA</td>
<td>M</td>
<td>Version</td>
<td>version</td>
<td>A version. This version must correspond to the specification of the extender.</td>
</tr>
</tbody>
</table>

Specifications for extenders (Blueprint, Declarative Services, etc.) should specify the values for these attributes. Extenders that provide such a capability should list the packages that they use in their specification in the uses directive of that capability to ensure class space consistency. Extenders can extend an extendee bundle even if that bundle does not require the extender unless the specification explicitly forbids this. For example an OSGi Blueprint Container could declare its capability with the following manifest header:

```manifest
Provide-Capability: osgi.extender;
osgi.extender="osgi.blueprint";
uses:=
"org.osgi.service.blueprint.container,org.osgi.service.blueprint.reflect";
version:Version="1.0"
```

A bundle that depends on a Blueprint Container could require such an extender with the following manifest header:

```manifest
Require-Capability: osgi.extender;
filter:="(&(osgi.extender=osgi.blueprint)(version>=1.0))"
```
The recommended way of using contracts is to create a contract bundle that provides the osgi.contract capability and imports the packages with their required version range. For example:

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

A contract may support multiple versions of a named contract. Such a contract must use a single capability for the contract name that specifies a list of all the versions that are supported. For example, the JavaServlet 3.1 contract capability would be specified with the following:

Provide-Capability: osgi.contract;
    osgi.contract=JavaServlet;
    version:List<Version>="2.5,3.0,3.1";
    uses:=
        "javax.servlet,
        javax.servlet.annotation,
        javax.servlet.descriptor,
        javax.servlet.http"
Export-Package:
    javax.servlet; version="3.1",
    javax.servlet.annotation; version="3.1",
    javax.servlet.descriptor; version="3.1",
    javax.servlet.http; version="3.1"

A client bundle that requires the Servlet 2.5 contract can then have the following manifest:

Require-Capability: osgi.contract;
    filter:="(&(osgi.contract=JavaServlet)(version>=2.5))",
Import-Package:
    javax.servlet, javax.servlet.http

The client bundle will be constrained by the contract's uses constraints and automatically gets the correct packages. In this example, no semantic versioning is used for the contract because the Servlet Specifications do not use semantic versioning (version 3.0 is backward compatible with 2.X).

In this model it is even possible to use the normally not recommended DynamicImport-Package header with a wild card since also this header is constrained by the uses constraints. However, using a full wildcard can also dynamically import packages that are not part of the contract. To prevent these unwanted dynamic imports, the exporter could include an attribute on the exports. For example:

Require-Capability: osgi.contract;
    filter:="(&(osgi.contract=JavaServlet)(version>=2.5))"
DynamicImport-Package:
    *;JavaServlet=contract

However, this model requires the exporter to specify an agreed attribute. The contract bundle does not require such coordination; it also allows the package exporters to reside in different and unrelated bundles.

A client bundle may only work with a specific range of contract versions. For example, a client may not work with JavaServlet with a version range of [3.0,3.1). In this case the following could be used:
osgi.service Namespace

The definition of the osgi.contract Namespace is in the following table and in the ContractNamespace class.

### Provide a pointer to the RFC180-defined contracts on the website.

Table 135.2 osgi.contract Namespace

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>M/O</th>
<th>Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>osgi.contract</td>
<td>CA</td>
<td>M</td>
<td>String</td>
<td>symbolic-name</td>
<td>A symbolic name for the contract.</td>
</tr>
<tr>
<td>version</td>
<td>CA</td>
<td>O</td>
<td>Version+</td>
<td>version</td>
<td>A list of versions for the contract. A contract that supports multiple versions must use a single capability with a version attribute that lists all versions supported.</td>
</tr>
<tr>
<td>uses</td>
<td>CD</td>
<td>O</td>
<td>String</td>
<td>package-name</td>
<td>For a contract, the standard uses clause is used to indicate which packages are part of the contract. The imports or exports of those packages link these packages to a particular version.</td>
</tr>
</tbody>
</table>

135.4 osgi.service Namespace

The Service Namespace is intended to be used for:

- Preventing a bundle from resolving if there is not at least one bundle that potentially can register a specific service.
- Providing a hint to the provisioning agent that the bundle requires a given service.
- Used as template for specifications like Blueprint and Declarative Services to express their provided and referenced services in the Repository model, see Repository Service Specification on page 531.

A bundle providing this capability indicates that it can register such a service with at least the given custom attributes as service properties. At resolve time this is a promise since there is no guarantee that during runtime the bundle will actually register such a service; clients must handle this with the normal runtime dependency managers like Blueprint, Declarative Services, or others.

See the following table and the ServiceNamespace class for this namespace definition.

Table 135.3 osgi.service Namespace

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>M/O</th>
<th>Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectClass</td>
<td>CA</td>
<td>M</td>
<td>List</td>
<td>qname</td>
<td>The fully qualified name of the object class of the service.</td>
</tr>
<tr>
<td>*</td>
<td>CA</td>
<td>O</td>
<td>*</td>
<td>*</td>
<td>Custom attributes that will be provided as service properties if they do not conflict with the service properties rules and are not private service properties (start with a full stop (’\u002E).</td>
</tr>
</tbody>
</table>

135.5 org.osgi.namespace.contract
Contract Namespace Package Version 1.0.
Bundles should not need to import this package at runtime since all the types in this package just contain constants for capability and requirement namespaces specified by the OSGi Alliance.

135.5.1 public final class ContractNamespace extends Namespace
Contract Capability and Requirement Namespace.
This class defines the names for the attributes and directives for this namespace. All unspecified capability attributes are of type String and are used as arbitrary matching attributes for the capability. The values associated with the specified directive and attribute keys are of type String, unless otherwise indicated.

Concurrency Immutable

135.5.1.1 public static final String CAPABILITY_VERSION_ATTRIBUTE = "version"
The capability attribute contains the Version of the specification of the contract. The value of this attribute must be of type Version.

135.5.1.2 public static final String CONTRACT_NAMESPACE = "osgi.contract"
Namespace name for contract capabilities and requirements.
Also, the capability attribute used to specify the name of the contract.

135.6 org.osgi.namespace.extender
Extender Namespace Package Version 1.0.
Bundles should not need to import this package at runtime since all the types in this package just contain constants for capability and requirement namespaces specified by the OSGi Alliance.

135.6.1 public final class ExtenderNamespace extends Namespace
Extender Capability and Requirement Namespace.
This class defines the names for the attributes and directives for this namespace. All unspecified capability attributes are of type String and are used as arbitrary matching attributes for the capability. The values associated with the specified directive and attribute keys are of type String, unless otherwise indicated.

Concurrency Immutable

135.6.1.1 public static final String CAPABILITY_VERSION_ATTRIBUTE = "version"
The capability attribute contains the Version of the specification of the extender. The value of this attribute must be of type Version.

135.6.1.2 public static final String EXTENDER_NAMESPACE = "osgi.extender"
Namespace name for extender capabilities and requirements.
Also, the capability attribute used to specify the name of the extender.

135.7 org.osgi.namespace.service
Service Namespace Package Version 1.0.

Bundles should not need to import this package at runtime since all the types in this package just contain constants for capability and requirement namespaces specified by the OSGi Alliance.

135.7.1 **public final class ServiceNamespace**

`extends Namespace`

Service Capability and Requirement Namespace.

This class defines the names for the attributes and directives for this namespace.

All unspecified capability attributes are of one of the following types:

- String
- Version
- Long
- Double
- List<String>
- List<Version>
- List<Long>
- List<Double>

and are used as arbitrary matching attributes for the capability. The values associated with the specified directive and attribute keys are of type String, unless otherwise indicated.

**Concurrency**
Immutable

135.7.1.1 **public static final String** `CAPABILITY_OBJECTCLASS_ATTRIBUTE = "objectClass"`

The capability attribute used to specify the types of the service. The value of this attribute must be of type `List<String>`.

A ServiceNamespace capability should express a uses constraint for all the packages mentioned in the value of this attribute.

135.7.1.2 **public static final String** `SERVICE_NAMESPACE = "osgi.service"`

Namespace name for service capabilities and requirements.

135.8 **References**

[1] Specification References
http://www.osgi.org/Specifications/Reference

OSGi Core, Chapter 6 Resource API Specification

OSGi Core, Chapter 8 Framework Namespaces Specification

OSGi Core, General Syntax Definitions

[5] Common Headers
OSGi Core, Chapter 3, Common Header Syntax
136 Resolver Service Specification

Version 1.0

136.1 Introduction

Today very few applications are self contained, the predominant development model is that applications are built from (external) components, which are often open source. Application developers add business logic, glue code, and assemble the diverse components into a resource that provides the desired capabilities when installed in an environment. Designing the assembly has long been a manual and error prone process, partly due to the complexity of external dependencies. Although the direct dependencies are often given, the largest number of dependencies are usually the transitive dependencies: the dependencies of the dependencies. Modern applications can end up with hundreds to thousands of external dependencies. Numbers that make tooling inevitable.

The OSGi framework is the first specification that provides a foundation for automating a significant part of this assembly process. The Requirement-Capability model defined in [3] Framework Resource API (Core) provides a dependency model that allows resources to express dependencies, constraints, and capabilities. If a resource’s constraints are met it provides capabilities that can satisfy further requirements. The OSGi dependency model is fully generic and is not limited to bundles. Resources can be bundles but also certificates, plugged in devices, etc.

Resolving transitive dependencies is a non-trivial process that requires careful design to achieve the required performance since the underlying problem is NP-complete. OSGi frameworks have always included such resolvers but these were built into the frameworks. They were not usable outside the framework for tooling, for example automatically finding the dependencies of a bundle that needs to be installed.

The number of dependencies is rapidly reaching a threshold where manual methods no longer can provide reliable results. This specification therefore provides the Resolver service, a service that can be the base for provisioning, deployment, build, and diagnostic tooling. The service can take a requirement and resolve it to a wiring of resources. For example, with cloud computing a new requirement can be translated into a new OSGi framework instance being started on a node and provisioned with the set of bundles that satisfy the given requirement. The OSGi Resolver service is intended be a corner stone of such an auto-provisioning tool.

However, the OSGi Resolver service is not limited to these higher end schemes. Build tools can use the Resolver to find components for the build path and/or run time environment and predict the results of installing a set of bundles on a target environment. The OSGi Resolver service is an essential part of a software model where applications are built out of independent components.

This specification is based on the concepts and API defined in the [3] Framework Resource API (Core), [2] Framework Wiring API (Core), and the [1] Framework Module Layer. These specifications are required reading for understanding this specification. This specification is for highly specialized use, it is not intended to be used in applications, the Resolver API is a low level service intended for system developers with deep knowledge of the OSGi module layer.

136.1.1 Essentials

- **Transitive** - From a requirement, find a consistent set of resources that satisfy that requirement.
- **Diagnostics** - Provide diagnostic information when no resolution can be found.
• **Scoped Repositories** - Allow the environment to control the repositories to use.
• **Build Tools** - Must be useful in establishing build and run time class paths.
• ** Provisioning** - Must be useful to find a set of bundles that can be installed in a system without running into unresolved dependencies.
• **OSGi** - Provide the semantics of all the OSGi namespaces, including the uses constraints.
• **API** - The API for the Resolver must provide the base for the Framework Bundle Wiring API.
• **Performant** - Enable highly performant implementations.
• **Frameworks** - Allow Frameworks to provide their resolver as a service.
• **Scalable** - Allow access to, and use of, very large repositories.

### 136.1.2 Entities

- **Environment** - A container or framework that can install resources and uses a Resolver to wire these resources.
- **Resolve Context** - An interface implemented by the management agent to provide the context of the resolution.
- **Wiring** - Represents the state of a resource’s wires, requirements, and capabilities in an environment.
- **Resolver** - A service that can find a set of wires that can be applied to an existing wiring state where a set of initial resources have all their mandatory requirements satisfied.
- **Wire** - Links requirement to a capability.
- **Resource** - An artifact with requirements that need to be provisioned and resolved to provide its capabilities.
- **Requirement** - A necessity for a given resource before it can provide its capabilities; expressed as a filter expression on the attributes of a capability in a given namespace.
- **Capability** - A set of attributes and directives defined by a namespace, describes what a resource provides when resolved.
- **Hosted Capability** - Pairs a resource with a capability to model hosting capabilities in another resource.
- **Namespace** - The type for a capability and requirement.
- **Resolution** - The result of a resolve operation.

**Figure 136.1** Class and Service overview
136.1.3 Synopsis

The Resolver service can find a complete and consistent set of transitive dependencies starting with an initial set of mandatory and optional resources. Such a set can be used to install resources in the environment from local or remote repositories. To use the Resolver service, the client must provide a ResolveContext object. This complex object represents the context of a resolution; it provides the initial resources (optional and mandatory), defines a number of namespaces specific policies, and provides the state of the environment.

A resolution can be based on an existing wiring in the environment, for example the current framework state. For the framework, the Resolve Context can find this existing state via the Framework Wiring API (Core). The Resolver must then satisfy all the requirements of the mandatory resources. The Resolver must always ask the Resolve Context to find additional capabilities for the unsatisfied requirements. A capability is always associated with a resource, which is subsequently associated with additional requirements. The final resolution must contain a set of resources that include the initial set of mandatory resources, has no unsatisfied mandatory requirements, and is consistent with the implied constraints. Otherwise it fails.

The Requirement-Capability model is fully generic but provides special semantics through the use of namespaces. The Resolver must implement the standard OSGi namespaces as described in Framework Wiring API (Core), which includes the uses constraints. Part of the semantics of the OSGi namespaces are delegated to the Resolve Context so that it can implement different policies. Singletons, ordering of capabilities, and matching are the responsibility of the Resolve Context; the Resolver never matches a requirement to a capability.

Requirements have an effective directive that indicates in what situations the requirement must be resolved. Also here, the Resolve Context decides if a particular requirement is effective or not. Only effective requirements are wired in the resolution.

Since capabilities are declared by resources that have additional requirements, the Resolver must create a set of resources where all transitive requirements are satisfied or fail with a Resolution Exception. This Resolution Exception can provide additional details why the resolution failed, if possible.

At the end of a successful resolve operation the Resolver returns a Map<Resource,List<Wire>>. These wires are a delta on the existing state, if any. The wires can then be used to provision missing resources or to provide diagnostic feedback.

136.2 The Resolve Context

Provisioning is the process of providing a framework with the necessary resources to allow it to operate according to set goals. In OSGi terms, this consists of installing bundles and ensuring that the configuration is set up correctly. With OSGi, bundles explicitly describe their capabilities and requirements as manifest headers. This can range from Export-Package (a capability) to a generic Provide-Capability header.

OSGi Frameworks have a resolving stage that ensures requirements are satisfied before a bundle is allowed to provide code to the shared space. As long as the requirements are not met, the bundle remains in the INSTALLED state and is thus prohibited from contributing capabilities. Once all the mandatory requirements are met, the bundle becomes RESOLVED. That is, a framework combines two decisions when it resolves bundles:

- Find a resolution based on the existing set of installed bundles.
- Move the bundles that have all their mandatory requirements satisfied to the RESOLVED state.

The Resolver service separates these two stages and thus allows a third party, the management agent, to define the environment of the resolution. A management agent can interact with the Resolver service
while it is searching for a resolution because the Resolver service calls back the management agent through a `ResolveContext` object. The Resolver service will therefore allow the management agent to handle more scenarios, better diagnostics, etc.

The Resolve Context is provided by the management agent, it is an abstract base class and must therefore be extended. It is a non-trivial class to implement since it is tightly coupled to the rules of the environment; it represents the policies of the management agent for that environment. For OSGi framework resolvers, the Resolve Context must understand and implement a part of the OSGi framework namespaces.

With the Resolver service, a management agent can try out resolutions before any bundle can see the result of such a resolution but it can also include extra bundles or other resources on demand. The Resolver service will also allow resolutions to be calculated for other frameworks.

For example, a management agent could use a Resolver service to find missing dependencies and install them on demand from a local directory with bundles. Such a Provisioner could have the following skeleton:

```java
public class Provisioner {
    File          bundles   = ...;
    Map<String,Resource> resources = ...;
    Resolver      resolver  = ...;
    BundleContext context   = ...;

    public void install(String location) {
        Resource resource = resources.get(location);
        if (resource == null) error(...);
        try {
            ResolveContextImpl rc = ...;
            rc.addMandatory(resource);
            Set<Resource> provision = resolver.resolve(rc).keySet();
            for (Resource rb : provision) {
                String location = getLocation(rb);
                Bundle bundle = context.installBundle(location);
                if (!isFragment(bundle))
                    bundle.start();
            }
        } catch(ResolutionException re) {
            ... // diagnostics
        } catch(BundleException be) {
            ... // diagnostics
        }
    }
}
```

### 136.2.1 Mandatory and Optional Resources

The Resolve Context provides all the parameters for the resolve operation, the Resolver does not maintain any state between invocations. The Resolve Context must therefore provide the mandatory and optional resources, which are essentially the input parameters to the resolve operation. The resolver must find a solution that includes at least the initial mandatory resources and should include the optional resources.
136.2.2 Finding Capabilities

The Resolve Context’s `findProviders(Requirement)` method must be implemented in such a way that it returns an ordered list of capabilities. The Resolver will treat the order of the capabilities as preferences, the first element is more preferred than a later element. The Resolver cannot guarantee that the wiring obeys this preference since there can be other constraints. However, a Resolver must use this preference order for simple cases and try to use it in more constrained situations.

The Resolver does not make any assumptions, this means that the `findProviders(Requirement)` method must do all the matching. Even though the Resolver gets the mandatory and optional resources it will not search these for capabilities to satisfy requirements. If the `findProviders(Requirement)` method does not search these resources then their capabilities will not be used. The same is true for the existing wiring state used.

Since this section describes the Resolver with respect to a provisioning agent, the set of resources is not limited to the installed set. That is, normally when a framework is resolved the Resolver only has to include installed resources. However, for a provisioning agent it is possible to retrieve external resources. The [Repository Service Specification](#) on page 531 provides access to resource repositories but a management agent is free to find capabilities by any alternative means.

For resolving an OSGi framework the specifications outlines a number of heuristics that guide the order of wiring bundles and packages:

1. A resource that is already resolved, that is, it is already wired
2. The highest version
3. The lowest bundle id

The Resolver can, and likely will, use the returned list to maintain its internal state during the resolve operation while trying out different potential solutions. It can add and remove capabilities at will. The returned list must therefore be mutable and not reused, after it is returned; it becomes owned by the Resolver. However, the Resolver can call back later with the `insertHostedCapability(List,HostedCapability)` method, giving back the returned list as the first parameter, see Insert Hosted Capabilities on page 646.

For example, assume that all possible resources in the previous example can be gotten with the `getSortedResources` method in the preferred resource order. This list contains all installed resources as well as any potentially installable resources. This list is sorted once on the given ordering criteria, this only has to be done once and not for each `findProviders(Requirement)` method invocation. The following code, which does not order by capability versions, could then be a simple skeleton for the `findProviders(Requirement)` method on the ResolveContextImpl inner class:

```java
public List<Capability> findProviders(Requirement requirement) {
    List<Capability> result = new ArrayList<Capability>();
    
    for ( Resource r : getSortedResources() )
        for ( Capability c : r.getCapabilities( null ) )
            if ( match( requirement, c ) )
                result.add( c );
    
    return result;
}
```

136.2.3 Matching

The `findProviders(Requirement)` method is responsible for only returning the capabilities that the management agent wants to be considered by the Resolver. Since the Resolver must never match any requirements to capabilities it allows the management agent to fully control the matching. However, in an OSGi environment the following matching behavior is expected:
• Requirements and capabilities must be in the same namespace.
• Only requirements and capabilities that have no effective directive or have the directive set to resolve should be considered.
• The requirement's filter must match the capability's attributes.
• If the namespace is an osgi.wiring.* namespace then the mandatory directive on the capability must be supported. Mandatory attributes are defined with a mandatory directive on a capability, they contain a list of attribute names. Each of these attributes must be used in the filter. Since the filter must be constructed from the corresponding manifest header it is sufficient to search the filter string with a regular expression that detects the usage of an attribute name.

The following example shows a skeleton match method that provides OSGi semantics:

```java
boolean match(Requirement r, Capability c) {
    if ( !r.getNamespace().equals( c.getNamespace() ) )
        return false;

    String effective = c.getDirectives().get("effective");
    if ( !(effective == null || effective.equals( "resolve") ) )
        return false;

    String f = r.getDirectives().get( "filter" );
    if ( f != null ) {
        Filter filter = context.createFilter( f );
        if ( !filter.matches( c.getAttributes() ) )
            return false;
    }

    if ( !c.getNamespace().startsWith( "osgi.wiring." ) )
        return true;

    String mandatory = c.getDirectives().get("mandatory");
    if ( mandatory == null )
        return true;

    List<String> attrs =
            Arrays.asList( mandatory.toLowerCase().split( "\\s*,\\s*" ) );

    Matcher m = FILTER_ASSERT_PATTERN.matcher(f == null ? "": f);
    while( m.find() )
        attrs.remove(m.group(1)); // the attribute name

    return mandatory.isEmpty();
}
```

### 136.2.4 Repositories

Resolving to provision a framework is different than a framework resolving itself. During provisioning remote repositories can be consulted to find external resources while the framework only resolves a limited set (the installed bundles). These repositories generally contain magnitudes more bundles than what is installed in a framework.

Repositories do not implement any namespace specific semantics and therefore do not understand any directives. Repositories only verify the requirement's filter (if any) against the capability's attributes. The Resolver expects the Resolve Context to do the namespace specific matching. The `Repository Service Specification` on page 531 provides the details for a service that abstracts a Requirement-Capability aware repository.
With such a repository service the `findProviders(Requirement)` method can be implemented as follows:

```java
List<Repository> repositories = new CopyOnWriteArrayList<Repository>();
void addRepository( Repository repository) { repositories.add(repository);}
void removeRepository(Repository repository){ repositories.remove(repository);}

class Repository {  // etc...

public List<Capability> findProviders( Requirement requirement) {
    List<Capability> result = new ArrayList<Capability>();
    // previous findProviders that searches the initial resources
    for ( Repository repository : repositories ) {
        Collection<Capability> capabilities = repository.findProviders(
            Collections.singleton( requirement ) ).get( requirement);
        for ( Capability c : capabilities )
            if ( match( requirement, c ) )
                result.add( c );
    }
    return result;
}
```

### 136.2.5 Existing Wiring State

The Resolver service always creates a list of wires that should be added to an existing state. To get the existing state, the `ResolveContext` interface specifies the `getWirings()` method. This method must return the existing state as a `Map<Resource,Wiring>`. A `Wiring` is an object that reflects the wired state of a resource in the environment. From this object, all declared and hosted capabilities and requirements can be found, including their wires if any. The Resolver needs this existing state to create a consistent resolution. For example, uses constraints require access to the existing state.

The Resolver service API is based on the generic Requirement-Capability model. This API is implemented by the OSGi framework to reflect its internal wiring, see [2] Framework Wiring API (Core). When the Resolver service is used for an OSGi framework then the Resolve Context can provide the existing wiring state based on the Framework Wiring API. The interfaces used in the `org.osgi.framework.wiring` package all extend their counterpart in the `org.osgi.resource` package (the generic model). For example, the `BundleCapability` interface extends the `Capability` interface.

The framework wiring API models all the power and complexities of the OSGi framework. One of those aspects is *removal pending*. Each installed bundle is represented by one or more bundle revisions. Each bundle revision is a Resource object but only one is the current bundle revision. During a resolve operation a framework can actually wire to the current bundle revision but is not forbidden to also select the pending removal bundle revisions. The Resolve Context must therefore decide if it provides only the current bundle revisions or all. The best policy solution in this case is to always refresh after a (batch) of install operations and only resolve when there are no pending-removal bundle revisions. However, certain management agents attempt to manage a system that is in this half-way state and will then be required to include the pending-removal revisions.

The following example code shows a possible implementation of the `getWirings()` method. It only uses the current wiring and ignores removal pending bundle revisions:

```java
public Map<Resource,Wiring> getWirings(){
    Map<Resource,Wiring> wirings = new HashMap<Resource,Wiring>();

    for ( Bundle b : context.getBundles() ) {
        BundleRevision revision = b.adapt( BundleRevision.class);
        if ( revision != null ) {
            // etc...
```

---

**Note:** The code snippets are provided as examples and may need to be adapted to fit into the specific context or environment they are intended for. The OSGi enterprise release specification version 1.0 is referenced throughout, indicating that the content is part of a formal specification for the OSGi technology.
Wiring wiring = revision.getWiring();
if ( wiring != null )
  wirings.put( revision, wiring );
}
return wirings;

136.2.6 Effective

The Resolver service is designed to work with OSGi frameworks but the scope is broader; its design allows many alternative usages. The effective directive on the capabilities and requirements is meant to allow requirements and capabilities to be effective in different phases. The default is resolve, the value for an OSGi framework resolving process. Bundles and other OSGi defined artifacts must declare their capabilities and requirements in the resolve effective time to be considered by an OSGi resolver.

However, Resolvers can use the effective directive to run the Resolver at other times than the standard OSGi framework resolve. For example, it could be possible to define an active time to resolve the service dependencies.

For this reason, the Resolver is designed to be agnostic for this directive, it will always ask the ResolveContext if a requirement is effective. It does this with the isEffective(Requirement) method. Since the Resolver service never matches requirements to capabilities it is also up to the Resolve Context to decide how to treat the effective directive. For an OSGi resolve operation, capabilities should also have an effective time value of resolve (or not set since resolve is the default).

To make requirements effective during the resolving of bundles it will be necessary to implement the isEffective(Requirement) method similar to:

```java
public boolean isEffective( Requirement requirement) {
  String e = requirement.getDirectives().get( "effective");
  return e==null || "resolve".equals( e );
}
```

136.2.7 Insert Hosted Capabilities

One of the complex aspects of resolving for an OSGi framework is handling fragments. For fragments, the declared capabilities are going to be hosted by their hosts. The Requirement and Capability objects have a getResource method that returns the associated resource. For hosted capabilities and requirements this must be the hosting resource and for others the declaring resource.

The HostedCapability interface defines the interface for allowing the hosting resource to be returned instead of the declaring resource. Since the Resolver service creates these Hosted Capabilities the Resolver needs a way to add them to the lists of capabilities returned from findProviders(Requirement). The Resolver service cannot add them itself since this list has a preference order, the Resolver service must therefore ask the Resolve Context to insert this new capability to allow the Resolve Context to maintain the desired order.

The Resolve Context must therefore implement an insertHostedCapability(List,HostedCapability) method. The given list must have been returned earlier from a findProviders(Requirement) method invocation. The Resolve Context must find the appropriate position to insert the HostedCapability object, insert it, and return the index of the inserted object.

It is the responsibility of the Resolve Context to find the proper position. In Finding Capabilities on page 643 it was discussed how the findProviders(Requirement) method must return an ordered list. The insertHostedCapability(List,HostedCapability) has that same responsibility.

The following example shows how the Hosted Capability is inserted based on the index of the hosted resource’s index in the sorted list of resources the management agent maintained. The example
iterates through the capabilities and compares the index of sorted resources to indicate preference. If it finds a capability from a resource that was later in the list of sorted resources then it inserts it at that position. A real implementation should also take the version of the capability into account.

```java
public int insertHostedCapability(
    List<Capability> caps, HostedCapability hc ) {
    List<Resource> resources = getSortedResources();
    int index = resources.indexOf( hc.getResource() );

    for ( int i =0; i < caps.size(); i++ ) {
        Capability c = caps.get( i );
        int otherIndex = resources.indexOf( c.getResource() );
        if ( otherIndex > index ) {
            caps.add( i, hc );
            return i;
        }
    }
    caps.add( hc );
    return caps.size()-1;
}
```

### 136.2.8 Fragments

Fragments are resources that have an `osgi.wiring.host` requirement that must match a capability from one or more host bundles. However, for example an Export-Package in a fragment must be merged with its attached hosts. These capabilities and requirements from namespaces that appear as if they come from the host bundle are called **hosted**.

Since fragments are not required by the host bundle, there will be no resource requiring the fragment bundles. The Resolver will therefore never ask the Resolve Context to provide fragments. However, fragments will require their hosts. A Resolver should attach any fragments available in a resolution to suitable hosts.

Since fragments will not be discovered from their hosts, it can be necessary to run the resolver twice in certain scenarios. The first resolve operation finds a resolution that is then used to find fragments that could be attached to the included bundles. These fragments can then be added to set of mandatory or optional resources for an additional resolve operation.

Fragments can of course also be found by the normal finding of capabilities.

### 136.2.9 Singleton Capabilities

A resource can be marked as a **singleton**. A singleton resource has the singleton directive set to true on the `osgi.identity` capability. A singleton resource **conflicts** with another singleton resource if:

- They have the same `osgi.identity`, and
- They have the same `type`, and
- They have a different or identical version.

This constraint is not enforced by the Resolver service to give more flexibility to management agents. The Resolve Context must ensure that it does not return capabilities from conflicting singleton resources from the `findProviders(Requirement)` method. When the Resolver is used with a limited set of resources then it is possible to enumerate all singletons ahead of time and use permutations. However, when repositories are used an incremental method works better because the scope is so much larger.

When the `findProviders(Requirement)` method is called for a requirement that selects a capability declared in a singleton then it is likely that repositories will return multiple versions of this single-
ton including the resource with the highest available version for conflicting resources. It is therefore possible to maintain a white list of singletons incrementally.

Once the `findProviders(Requirement)` method has created a result list, it is possible to prune this list of conflicting singletons. This can be an incremental process, the first time a singleton is encountered in such a list of capabilities the highest version can be selected as the singleton. Other singletons that are in that list or come in other invocations of `findProviders(Requirement)` can then no longer provide capabilities. For example:

```java
Map<String, Resource> whitelist = new HashMap<String, Resource>();

void prune( List<Capability> list ) {
    Map<String, Resource> singletons = new HashMap<String, Resource>();

    for ( Capability c : list ) {
        Resource r = c.getResource();
        Version now = getVersion( r );
        String identity = getIdentity( r );

        if ( isSingleton( r ) && !whitelist.containsKey( identity ) ) {
            Resource selected = singletons.get( identity );
            if ( selected == null )
                singletons.put( identity, r );
            else {
                Version old = getVersion( selected );
                if ( now.compareTo( old ) > 0 )
                    singletons.put( identity, r );
            }
        }
    }

    this.whitelist.putAll( singletons );

    for ( Iterator<Capability> i=list.iterator(); i.hasNext(); ) {
        Capability c = i.next();
        Resource r = c.getResource();
        String identity = getIdentity( r );
        Resource selected = this.whitelist.get( identity );
        if ( selected != null && !selected.equals( r ))
            i.remove();
    }
}
```

### 136.2.10 Diagnostics

The Resolve service throws a `ResolutionException` when the resolve operation cannot find a solution. This Exception provides the standard human readable message. However, there is also the `getUnresolvedRequirements()` method. With this method it is possible to find what requirements could not be matched. Though this is very useful in many cases it must be realized that resolving is a complicated process. It is not possible to establish the exact set of missing requirements because any unresolved resolution can have many, different, sets of unresolved requirements. This is an intrinsic part of the resolution problem. There is also no guarantee that providing capabilities that satisfy these requirements will give a successful resolution. There could still be other constraints that cannot be satisfied. In general, the Resolve Context can already detect any unresolved mandatory requirements when it cannot find a matching capability in the `findProviders(Requirement)` method.

That said, the `getUnresolvedRequirements()` can often point to a potential solution.
136.2.11 Complexity

Implementing a Resolve Context is a non-trivial task that requires extensive knowledge of the OSGi framework, especially the module layer. Though this section contains numerous code examples, they are not sufficient to implement a real Resolve Context since this would require too much code for a specification.

136.3 Resolver Service

The Resolver service is an interface to a generic constraint solver based on the Require-Capability model defined in [3] Framework Resource API (Core). This model defines a constraint-solving language that is used by the Framework, see [1] Framework Module Layer, to create the mesh of class loaders. However, the Resolver service has been designed to be useful in solving other types of constraint problems.

The task of the Resolver is to find a resolution. The resolve method returns a delta on an existing wiring state. The total of existing wiring state and the applied delta is the resolution. The delta is a set of wires between requirements and capabilities.

136.3.1 Variables

The resolve(ResolveContext) method uses a Resolve Context to provide the context and parameters of the resolution. During the resolution process the Resolver service can callback the Resolve Context to retrieve the following information:

\[\begin{align*}
R_m & : \text{Collection<Resource>} \quad \text{getMandatoryResources()} \\
R_o & : \text{Collection<Resource>} \quad \text{getOptionalResources()} \\
C_{env} & : \text{Map<Requirement,List<Capability>>} \quad \text{Combined answers from the findProviders(Requirement) method} \\
Q_{eff} & : \text{Collection<Requirement>} \quad \text{Set of effective requirements as defined by the isEffective(Requirement) method} \\
X & : \text{Map<Resource,Wiring>} \quad \text{An existing Wiring state, getWirings()} \\
D & : \text{Map<Resource,List<Wire>>} \quad \text{The resolution, a delta on the existing state}
\end{align*}\]

The resolve(ResolveContext) method returns a resolution D that is a delta on the existing Wiring state X. It is up to the Resolve Context to ensure that the delta D is installed. In for example the OSGi framework the framework hooks can be used to guide the framework’s resolving process.

136.3.2 Resolving

The goal of the Resolver is to provide a set of wires between a set of resolved resources. A resource is resolved when all its mandatory and effective requirements are satisfied by capabilities from resolved resources. A Resolver must not return wires between resources that have unsatisfied mandatory requirements.

A mandatory requirement has a resolution directive that is not set or that is set to mandatory. The effectiveness of a requirement is defined by the Resolve Context, a Resolver service must call the isEffective(Requirement) method to establish if a requirement is effective for a given resolve operation. A Resolver must never create a wire from a requirement that is not effective.

To find a resolution, the Resolver must use the Resolve Context to find candidate capabilities for the requirements declared in the resources that it needs to resolve. A candidate capability is a capability that satisfies the requirement. From the perspective of the Resolver service, a requirement is satis-
fied by a capability when that capability is returned from the `findProviders(Requirement)` method. A Resolver has no alternative way to find out if a requirement is satisfied by a capability and must therefore not make any assumptions about matching. Any matching rules like for example the `osgi.wiring.* mandatory` directive must be implemented by the Resolve Context. A Resolve Context must always return the same capabilities during a resolving operation when given the same requirement.

Since the resolver cannot match requirements and capabilities the Resolve Context must return capabilities for all possible resources, this must include:

- The given mandatory resources $R_m$
- The given optional resources $R_o$
- The existing Wiring state $X$

It can include additional resources that were indirectly returned through the `findProviders(Requirement)` method.

The existing wiring $X$ provides an existing set of constraints that the Resolver service must abide by. It can get this state with the `getWirings()` method. The purpose of the existing state is to ensure that any wires are compatible with existing wiring. For an OSGi framework it is crucial that the uses constraints are maintained for the resolution, see [1] Framework Module Layer.

The Resolver service can wire new requirements to existing capabilities but it can never create wires for an existing requirement from the existing wiring.

If the Resolver service attaches a hosted resource like a fragment, and thereby needs to add new `HostedCapability` objects to a previously returned list from `findProviders(Requirement)` then it must call the Resolve Context's `insertHostedCapability(List,HostedCapability)` method.

Fragments can be attached from resolved resources or from resources that declare the capabilities returned from `findProviders(Requirement)`, that is, $C_{env}$.

This specification does not define the detailed rules and constraints associated with resolving; these rules are defined in their namespaces. An OSGi Resolver service must support at least all namespaces defined in [4] Framework Namespace (Core) except for the following directives:

- `mandatory` - Mandatory attributes on the `osgi.wiring.*` namespaces must be implemented by the Resolve Context. The Resolve Context should not return capabilities from `findProviders(Requirement)` unless the rules of the OSGi mandatory directive are followed.
- `singleton` - Singletons are not implemented by the Resolver, the Resolve Context must not return capabilities from `findProviders(Requirement)` from conflicting singleton resources.
- `effective` - The Resolve Context decides what requirements are effective in the `isEffective(Requirement)` method.

A Resolver service must support the uses constraints and any applicable rule defined in the [1] Framework Module Layer for the `osgi.wiring.*` namespaces.

The Resolver must return a delta wiring to the existing state (potentially empty) or throw an Exception. The resolution:

- Must contain all mandatory resources $R_m$ as provided by `getMandatoryResources()`.
- Must have all resources resolved.
- Must have no wired capabilities that are declared or hosted in resources that are not resolved.
- Should include optional resources $R_o$ as provided by `getOptionalResources()`.

### 136.3.3 Resolution Exception

If the Resolver cannot find a solution or it runs into problems then it must throw a Resolution Exception, which is a Runtime Exception.
The ResolutionException provides the getUnresolvedRequirements() method. If the resolution failed then it is possible that this was caused because it failed to find matches for certain requirements. The information in this method can be very helpful to find a solution that will work, however, there are a number of caveats.

Resolving is an NP-complete problem. For these problems there exists no algorithm that can infer a solution from the desired outcome. Therefore, the Resolver tries a potential solution and if that solution does not match the constraints it will backtrack and attempt another solution. An unavoidable aspect of such solutions is that it is impossible to pin-point a single failure point if the algorithm fails to find a solution, in general the algorithm gives up after having exhausted its search space. However, during its search it might have been very close to a solution, for example it only missed a single requirement, but its final failure missed many requirements.

The implication is that the reported missing requirements neither give a guarantee for a resolution when satisfied nor indicate that this is the smallest set of missing requirements.

Therefore, getUnresolvedRequirements() is intended for human consumption and not for automated solutions.

### 136.4 Security

#### 136.4.1 Resolving

The Resolver service is a pure function that has no state. The Resolve Context maintains the state and is therefore the actor that requires most permissions. In general, it will require access to the Wiring API and Repositories.

Since the Resolver requires no external access it does not have to be a trusted service. Resolve Contexts that support security must ensure that the callbacks are executed in a privileged block.

#### 136.4.2 Minimum Implementation Permissions

- PackagePermission[org.osgi.service.resolver,IMPORT]
- ServicePermission[...Resolver, REGISTER ]

#### 136.4.3 Minimum Using Permissions

- PackagePermission[org.osgi.service.repository,IMPORT]
- PackagePermission[org.osgi.service.resolver,IMPORT]
- PackagePermission[org.osgi.resource,IMPORT]
- PackagePermission[org.osgi.framework.wiring,IMPORT]
- PackagePermission[org.osgi.framework.namespaces,IMPORT]
- ServicePermission[...Resolver, GET ]
  - ... likely needs AdaptPermissions and ServicePermission[...Repository,GET]

### 136.5 org.osgi.service.resolver

Resolver 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. 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.resolver; version="[1.0,2.0)"
Example import for providers implementing the API in this package:
Import-Package: org.osgi.service.resolver; version="[1.0,1.1)"

136.5.1 Summary

- HostedCapability - A capability hosted by a resource.
- ResolutionException - Indicates failure to resolve a set of requirements.
- ResolveContext - A resolve context provides resources, options and constraints to the potential solution of a resolve operation.
- Resolver - A resolver service resolves the specified resources in the context supplied by the caller.

136.5.2 public interface HostedCapability
extends Capability

A capability hosted by a resource.

A HostedCapability is a Capability where the getResource() method returns a Resource that hosts this Capability instead of declaring it. This is necessary for cases where the declaring Resource of a Capability does not match the runtime state. For example, this is the case for fragments attached to a host. Most fragment declared capabilities and requirements become hosted by the host resource. Since a fragment can attach to multiple hosts, a single capability can actually be hosted multiple times.

Concurrency Thread-safe
Provider Type Consumers of this API must not implement this type

136.5.2.1 public Capability getDeclaredCapability()

- Return the Capability hosted by the Resource.

Returns The Capability hosted by the Resource.

136.5.2.2 public Resource getResource()

- Return the Resource that hosts this Capability.

Returns The Resource that hosts this Capability.

136.5.3 public class ResolutionException
extends Exception

Indicates failure to resolve a set of requirements.

If a resolution failure is caused by a missing mandatory dependency a resolver may include any requirements it has considered in the resolution exception. Clients may access this set of dependencies via the getUnresolvedRequirements() method.

Resolver implementations may extend this class to provide extra state information about the reason for the resolution failure.

136.5.3.1 public ResolutionException(String message, Throwable cause, Collection<Requirement> unresolvedRequirements)

message The message.
cause The cause of this exception.
unresolvedRequirements The unresolved mandatory requirements from mandatory resources or null if no unresolved requirements information is provided.

- Create a ResolutionException with the specified message, cause and unresolved requirements.
136.5.3.2 public ResolutionException(String message)

message The message.

Create a ResolutionException with the specified message.

136.5.3.3 public ResolutionException(Throwable cause)

cause The cause of this exception.

Create a ResolutionException with the specified cause.

136.5.3.4 public Collection<Requirement> getUnresolvedRequirements()

Return the unresolved requirements, if any, for this exception.

The unresolved requirements are provided for informational purposes and the specific set of unresolved requirements that are provided after a resolve failure is not defined.

Returns A collection of the unresolved requirements for this exception. The returned collection may be empty if no unresolved requirements information is available.

136.5.4 public abstract class ResolveContext

A resolve context provides resources, options and constraints to the potential solution of a resolve operation.

Resolve Contexts:

- Specify the mandatory and optional resources to resolve. The mandatory and optional resources must be consistent and correct. For example, they must not violate the singleton policy of the implementer.
- Provide capabilities that the Resolver can use to satisfy requirements via the findProviders(Requirement) method
- Constrain solutions via the getWirings() method. A wiring consists of a map of existing resources to wiring.
- Filter requirements that are part of a resolve operation via the isEffective(Requirement).

A resolver may call the methods on the resolve context any number of times during a resolve operation using any thread. Implementors should ensure that this class is properly thread safe.

Except for insertHostedCapability(List, HostedCapability), the resolve context methods must be idempotent. This means that resources must have constant capabilities and requirements and the resolve context must return a consistent set of capabilities, wires and effective requirements.

Concurrency Thread-safe

136.5.4.1 public ResolveContext()
The (optional) method to add additional Capabilities to maintain priority order. In general, this is necessary when the Resolver uses Capabilities declared in a Resource but that must originate from an attached host.

Each returned Capability must match the given Requirement. This means that the filter in the Requirement must match as well as any namespace specific directives. For example, the mandatory attributes for the osgi.wiring.package namespace.

Returns A list of Capability objects that match the specified requirement.

136.5.4.3 public Collection<Resource> getMandatoryResources()

Return the resources that must be resolved for this resolve context.

The default implementation returns an empty collection.

Returns A collection of the resources that must be resolved for this resolve context. May be empty if there are no mandatory resources. The returned collection may be unmodifiable.

136.5.4.4 public Collection<Resource> getOptionalResources()

Return the resources that the resolver should attempt to resolve for this resolve context. Inability to resolve one of the specified resources will not result in a resolution exception.

The default implementation returns an empty collection.

Returns A collection of the resources that the resolver should attempt to resolve for this resolve context. May be empty if there are no optional resources. The returned collection may be unmodifiable.

136.5.4.5 public abstract Map<Resource,Wiring> getWirings()

Return the wirings for existing resolved resources.

For example, if this resolve context is for an OSGi framework, then the result would contain all the currently resolved bundles with each bundle’s current wiring.

Multiple calls to this method for this resolve context must return the same result.

Returns The wirings for existing resolved resources. The returned map is unmodifiable.

136.5.4.6 public abstract int insertHostedCapability(List<Capability> capabilities,HostedCapability hostedCapability)

capabilities The list returned from findProviders(Requirement). Must not be null.

hostedCapability The HostedCapability to insert in the specified list. Must not be null.

Add a HostedCapability to the list of capabilities returned from findProviders(Requirement).

This method is used by the Resolver to add Capabilities that are hosted by another Resource to the list of Capabilities returned from findProviders(Requirement). This function is necessary to allow fragments to attach to hosts, thereby changing the origin of a Capability. This method must insert the specified HostedCapability in a place that makes the list maintain the preference order. It must return the index in the list of the inserted HostedCapability.

Returns The index in the list of the inserted HostedCapability.

136.5.4.7 public abstract boolean isEffective(Requirement requirement)

requirement The Requirement to test. Must not be null.

Test if a given requirement should be wired in the resolve operation. If this method returns false, then the resolver should ignore this requirement during the resolve operation.

The primary use case for this is to test the effective directive on the requirement, though implementations are free to use any effective test.

Returns true if the requirement should be considered as part of the resolve operation.
136.5.5  public interface Resolver

A resolver service resolves the specified resources in the context supplied by the caller.

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

136.5.5.1 public Map<Resource,List<Wire>> resolve(ResolveContext context) throws ResolutionException

context The resolve context for the resolve operation. Must not be null.

Resolve the specified resolve context and return any new resources and wires to the caller.

The resolver considers two groups of resources:

- Mandatory - any resource in the mandatory group must be resolved. A failure to satisfy any mandatory requirement for these resources will result in throwing a ResolutionException
- Optional - any resource in the optional group may be resolved. A failure to satisfy a mandatory requirement for a resource in this group will not fail the overall resolution but no resources or wires will be returned for that resource.

The resolve method returns the delta between the start state defined by ResolveContext.getWirings() and the end resolved state. That is, only new resources and wires are included.

The behavior of the resolver is not defined if the specified resolve context supplies inconsistent information.

Returns The new resources and wires required to satisfy the specified resolve context. The returned map is the property of the caller and can be modified by the caller.

Throws ResolutionException – If the resolution cannot be satisfied.

136.6 References

[1] Framework Module Layer
OSGi Core, Chapter 3 Module Layer

[2] Framework Wiring API (Core)
OSGi Core, Chapter 7 Bundle Wiring API Specification

[3] Framework Resource API (Core)
OSGi Core, Chapter 6 Resource API Specification

[4] Framework Namespace (Core)
OSGi Core, Chapter 8 Framework Namespaces Specification
137  REST Management Service Specification

Version 1.0

137.1  Introduction

Cloud computing is a continuing trend in the IT industry. Due to its service model which embraces dynamism as opposed to masking it, OSGi appears to be an ideal base for building scalable and dependable applications for the cloud where changes in the deployment, network topology, and service availability are the norm rather than the exception. One of the possible scenarios for OSGi to be successfully applied to cloud computing is using it in a Platform as a Service (PaaS) spirit. Users write their bundles and can deploy them to their own OSGi instance running in the cloud. This, however, requires the platform provider to expose the OSGi management API to the end user and make them available through a network protocol. One of the popular approaches in cloud computing to remote communication is the use of RESTful web services.

Representational State Transfer (REST) is the architectural style of the world wide web. It can be described as a set of constraints that govern the interactions between the main components of the Internet. Recently, REST style interaction has gained popularity as a architecture for web services (RESTful web services), mainly to overcome the perceived complexity and verbosity of SOAP-based web services. This specification describes a REST interface for framework management, client-side Java and JavaScript APIs, and an extension mechanism through which other bundles can contribute their own RESTful management APIs and make them discoverable by clients.

137.1.1  Essentials

- **Client-Server** A separation of concern between the entity responsible for the user interaction (client) and the other entity (server) responsible for data storage. For instance, in the original world wide web the browser is the client rendering and presenting the content delivered by one or more web servers. As a result, web content becomes more portable and content providers more scalable.

- **Stateless** State is entirely kept at the client side. Therefore, every request must contain all state required for the server to accomplish the transaction and deliver content. The main rationale behind this design constraint is to again improve the scalability since in a pure stateless design the server resources are not burdened with maintaining any client state. Another perceived advantage is that the failure models of stateless interactions is simpler and fault tolerance easier to achieve.

- **Cacheable** Content marked as cacheable can be temporarily stored and used to immediately answer future equivalent requests and improve efficiency and reduce network utilization and access latencies. Due to the end-to-end principle, caches can be placed where necessary, e.g., at the client (forward-proxy), or at the server side (backward-proxy). Content marked as non-cacheable must be freshly retrieved with every request even in the presence of caches.

- **Layered** Layering introduces natural boundaries to coupling since every layer only accesses the services provided by the lower layer and provides services to the next higher layer.
• **Uniform Interface** Generality of component interfaces provides a natural decoupling of implementation and interface. REST furthermore encourages the separation of identifiable resources (addressing) and their representation (content delivery).

### 137.2 Interacting with the REST Management Service

The REST Management Service is not a traditional OSGi service and it does not even appear in the service registry. Its purpose is to expose a management interface to clients which can perform operations on the framework through a network connection. Therefore, it is ideally suited for situations where the user of an OSGi framework does not have direct access to the machine it is running on, a typical situation in Infrastructure as a Service (IaaS) or Platform as a Service (PaaS). However, even in other domains having a lightweight and easily accessible management solution can be of benefit, e.g., for embedded devices. The advantage of REST is that it uses HTTP and therefore does usually not interfere with firewalls. Furthermore, the REST format is easily embeddable into client-side scripting technologies like JavaScript and can be consumed in web browsers.

Much of the value of the REST Management Service lies in client-side libraries which can use the REST protocol and interact with the OSGi framework through the Management Service. Therefore, this specification contains API for two clients, a Java Client API and a JavaScript Client API.

### 137.2.1 Resource Identifier Overview

The REST Management Service comprises of a set of resources that can be retrieved and in some cases also modified through REST requests. These resources need to be made available under well-defined paths so that clients can interact with them. As the initial entry point a client receives a URL to the REST Management Service. This can be done, e.g., as part of the creation of a cloud-based OSGi
framework, and the precise mechanism would be proprietary to the cloud platform used. Relative to this URL the client can access the resources through the following resource identifiers:

- framework
- framework/state
- framework/startlevel
- framework/bundles
- framework/bundles/representations
- framework/bundle/{bundleid}
- framework/bundle/{bundleid}/state
- framework/bundle/{bundleid}/startlevel
- framework/bundle/{bundleid}/header
- framework/services
- framework/services/representations
- framework/service/{serviceid}
- framework/bundle/0/state

framework/bundle/0/state is an alias for framework/state

The bundles, bundles/representations, services, and services/representations resources allow the use of a query parameter which specifies a filter to restrict the result set. The filter expression follows the Core Specifications Framework Filter Syntax[100]. Filters on services are matched against the service attributes. The query parameter is of the form: framework/services?filter=ldap-filter

Filters on bundles are matched against the attributes of capabilities in the respective namespaces. Filters on bundles have the form: framework/bundles?namespace1=ldap-filter1&namespace2=ldap-filter2... A missing namespace declaration implies the IdentityNamespace (“osgi.identity”).

### 137.2.2 Content Type Matching

Resources can present themselves through different representation variants. For instance, an implementation of the REST Management Service could support both a JSON representation and an XML representation of the same resource. Matching the clients capabilities to understand certain representation formats with the servers supported formats follows the typical HTTP pattern of content negotiation and requires the client to set corresponding HTTP Accept headers for supported formats in the form of their media types. This specification describes the format and media types for representations in JSON and XML format in Representations on page 665.

Implementations of the REST Management Service offering different variants of representations must return the best matching variant based on the HTTP accept header. In addition, they must respect the file extensions defined for the different media types as specified in the respective IETF RFC (e.g., “.xml” as specified in IETF RFC 3032 and “.json” as specified in IETF RFC 4627). If a file extension is appended to the resource, an implementation must return the variant mandated by the file extension provided that it supports this content type.

### 137.2.3 Versioning and Interoperability

All representations described in this specification have version 1. Future versions of the representations must contain the version number in their content type (e.g., application/org.osgi.bundlesV2) to allow legacy clients to explicitly request an older version of the representation by setting their accept header accordingly.

Clients must understand all attributes described in this documents and may ignore any further attribute that a specific implementation of the REST Management Service might add.
137.3 Resources

The framework and its state is mapped to a set of different resources. Each resource is accessible through a resource identifier, as summarized in Resource Identifier Overview on page 658.

137.3.1 Framework Startlevel Resource

The startlevel resource represents the active start level of the framework. It supports the GET and PUT requests.

137.3.1.1 GET

The GET request retrieves a Framework Startlevel Representation from the REST management service. The request can return the following status codes:

- 200 (OK): the request has been served successfully and the body of the response is a startlevel representation.
- 406 (NOT ACCEPTABLE): the REST management service does not support any of the requested representations.

137.3.1.2 PUT

The PUT request sets the target framework startlevel. The body of the request needs to be a Framework Startlevel Representation. The request can return the following status codes:

- 204 (NO CONTENT): the request was received and valid. The framework will asynchronously start to adjust the framework startlevel until the target startlevel has been reached.
- 415 (UNSUPPORTED MEDIA TYPE): the request had a media type that is not supported by the REST management service.
- 400 (BAD REQUEST): the REST management service received an IllegalArgumentException when trying to adjust the framework startlevel, e.g., because the requested startlevel was zero or negative.

137.3.2 Bundles Resource

The bundles resource represents the list of all bundles installed on the managed framework. It supports the GET request and two syntactically different forms of POST requests which are used to install new bundles to the framework.

137.3.2.1 GET

The GET request retrieves a Bundle List Representation from the REST management service. The request can return the following status codes:

- 200 (OK): the request has been served successfully and the body of the response is a bundle list representation.
- 406 (NOT ACCEPTABLE): the REST management service does not support any of the requested representations.

137.3.2.2 POST with Location String

The POST request installs a new bundle to the managed framework and thereby logically appends it to the bundles resource. The new bundle to be installed is referenced by a location string which is passed as the body of the request. In order to disambiguate the request from the other form of POST, the content type must be set to text/plain. In practice, the location string is usually a URL. Since the framework will use the location retrieving the physical bundle, it needs to be accessible from the remotely managed framework and not necessarily from the managing client.
The management service implementation must check if the result of the install request matches the requested bundle since the OSGi framework will return an existing bundle object as the return value of an install call if there was already one with the same location string installed. One way of doing it is comparing the last modification timestamp. A detected collision is indicated to the requesting clients through an error code 409.

The response for the POST request is a local URI denoting path of the newly installed bundle relative to the REST management service root URL. The request can return the following status codes:

- 200 (OK): the bundle has been successfully installed and the body of the response contains the URI.
- 400 (BAD REQUEST): the REST management service received a BundleException when trying to install. The body of the message is a Bundle Exception Representation describing the reason why the installation did not succeed.
- 409 (CONFLICT): there is already a bundle installed with the same location string.

### POST with Bundle

This variant of the POST request uploads the bundle as the body of the request. The media type of the request should be set to vnd.osgi.bundle which must be supported by all REST management services. Implementations are free to accept other media types for this request with the exception of text/plain. For instance, they can opt to additionally support application/zip or application/x-jar.

Clients should use the HTTP Content-Location field to set a bundle location. If no content location is given, REST management service implementations must generate a unique location string in order to avoid unintended collisions between unrelated bundles.

The response for the POST request is a local URI denoting path of the newly installed bundle relative to the REST management service root URL. The request can return the following status codes:

- 200 (OK): the bundle has been successfully installed and the body of the response contains the URI.
- 400 (BAD REQUEST): the REST management service received a BundleException when trying to install. The body of the message is a Bundle Exception Representation describing the reason why the installation did not succeed.
- 409 (CONFLICT): there is already a bundle installed with the same location string.

### GET of the Representations

The bundles resource returns a list of the URIs of all bundles installed on the framework. For clients interested in all bundles there is also the possibility to retrieve the bundle representation of each installed bundle with a single request through the bundles/representations resource.

The body of the response is a Bundle Representations List Representation. The request can return the following status codes:

- 200 (OK): the request has been served successfully and the body of the response is a bundle list representation.
- 406 (NOT ACCEPTABLE): the REST management service does not support any of the requested representations.

### Bundle Resource

The bundle resource represents a single, distinct bundle in the system. Hence, it has to be qualified by a bundle id. The resource supports the GET, two variants of PUT, and the DELETE requests.

#### GET

The GET request retrieves a Bundle Representation from the REST management service. The request can return the following status codes:
• 200 (OK): the request has been served successfully and the body of the response is a bundle representation.
• 404 (NOT FOUND): there is not bundle with the given bundle id.
• 406 (NOT ACCEPTABLE): the REST management service does not support any of the requested representations.

137.3.3.2 PUT with Location String
The PUT request updates the bundle with a new version, referenced by a location string which is passed as the body of the request. In order to disambiguate the request from the other form of PUT, the content type must be set to text/plain. The same rationale applies as for POST with Location String and POST with Bundle on page 661, if a location string is given it must point to a location reachable by the managed framework. If no location string is passed as the body of the request, the framework will perform an update based on the existing bundle’s location string. The request can return the following status codes:

• 204 (NO CONTENT): the request was received and valid and the framework has issued the update.
• 400 (BAD REQUEST): the REST management service received a BundleException when trying to update. The body of the message is a Bundle Exception Representation describing the reason why the update did not succeed.
• 404 (NOT FOUND): there is not bundle with the given bundle id.

137.3.3.3 PUT with Bundle
The PUT request updates the bundle with a new version, uploaded as the body of the request. The media type of the request should be set to vnd.osgi.bundle which must be supported by all REST management services. Implementations are free to accept other media types for this request with the exception of text/plain. For instance, they can opt to additionally support application/zip or application/x-jar. The request can return the following status codes:

• 204 (NO CONTENT): the request was received and valid and the framework has issued the update.
• 400 (BAD REQUEST): the REST management service received a BundleException when trying to update. The body of the message is a Bundle Exception Representation describing the reason why the update did not succeed.
• 404 (NOT FOUND): there is not bundle with the given bundle id.

137.3.4 DELETE
The DELETE request uninstalls the bundle from the framework. The request can return the following status codes:

• 204 (NO CONTENT): the request was received and valid and the framework has uninstalled the bundle.
• 400 (BAD REQUEST): the REST management service received a BundleException when trying to uninstall. The body of the message is a Bundle Exception Representation describing the reason why the uninstallation did not succeed.
• 404 (NOT FOUND): there is not bundle with the given bundle id.

137.3.4 Bundle State Resource
The bundle state resource represents the internal state of an installed bundle qualified through its bundle id. It supports the GET and PUT requests.
137.3.4.1 GET

The GET request retrieves a Bundle State Representation from the REST management service. The request can return the following status codes:

- 200 (OK): the request has been served successfully and the body of the response is a bundle state representation.
- 404 (NOT FOUND): there is no bundle with the given bundle id.
- 406 (NOT ACCEPTABLE): the REST management service does not support any of the requested representations.

137.3.4.2 PUT

The PUT request sets the target state for the given bundle. This can, e.g., be state=32 for transitioning the bundle to started, or state=4 for stopping the bundle and transitioning it to resolved. The body of the request needs to be a Bundle State Representation. Not all state transitions are valid. The body of the response is the new Bundle State Representation. The request can return the following status codes:

- 200 (OK): the request was received and valid. The framework has performed a state change and the new bundle state is contained in the body.
- 400 (BAD REQUEST): the REST management service received a BundleException when trying to perform the state transition. The body of the message is a Bundle Exception Representation describing the reason why the operation did not succeed.
- 402 (PRECONDITION FAILED): the requested target state is not reachable from the current bundle state.
- 404 (NOT FOUND): there is no bundle with the given bundle id.
- 415 (UNSUPPORTED MEDIA TYPE): the request had a media type that is not supported by the REST management service.

137.3.5 Bundle Header Resource

The bundle header resource represents manifest header of a bundle which is qualified by its bundle id. It can only be read through a GET request.

137.3.5.1 GET

The GET request retrieves a Bundle Header Representation from the REST management service. The request can return the following status codes:

- 200 (OK): the request has been served successfully and the body of the response is a bundle header representation.
- 404 (NOT FOUND): there is no bundle with the given bundle id.
- 406 (NOT ACCEPTABLE): the REST management service does not support any of the requested representations.

137.3.6 Bundle Startlevel Resource

The bundle startlevel resource represents the start level of the bundle qualified by its bundle id. It supports the GET and PUT requests.

137.3.6.1 GET

The GET request retrieves a Bundle Startlevel Representation from the REST management service. The request can return the following status codes:

- 200 (OK): the request has been served successfully and the body of the response is a bundle startlevel representation.
- 404 (NOT FOUND): there is no bundle with the given bundle id.
• 406 (NOT ACCEPTABLE): the REST management service does not support any of the requested representations.

**PUT**

The PUT request sets the target bundle startlevel. The body of the request needs to be a *Bundle Startlevel Representation*. The request can return the following status codes:

• 200 (OK): the request was received and valid. The REST management service has changed the bundle startlevel according to the target value. The body of the response is the new bundle startlevel representation.
• 400 (BAD REQUEST): either the target startlevel state involved invalid values, e.g., a startlevel smaller or equal to zero and the REST management service got an IllegalArgumentException, or the REST management service received a BundleException when trying to perform the startlevel change. In the latter case, the body of the message is a *Bundle Exception Representation* describing the reason why the operation did not succeed.
• 404 (NOT FOUND): there is not bundle with the given bundle id.
• 415 (UNSUPPORTED MEDIA TYPE): the request had a media type that is not supported by the REST management service.

**Services Resource**

The services resource represents the set of all services available on the framework, optionally constrained by a filter expression. It is read-only and therefore only supports the GET request.

**GET**

The GET request retrieves a *Service List Representation* from the REST management service. The request can return the following status codes:

• 200 (OK): the request has been served successfully and the body of the response is a service list representation.
• 400 (BAD REQUEST): the provided filter expression was not valid.
• 406 (NOT ACCEPTABLE): the REST management service does not support any of the requested representations.

**GET of the Representations**

The services resource returns a list of the URIs of all services registered on the framework. For clients interested in all services there is also the possibility to retrieve the service representation of each available service with a single request through the `services/representations` resource. The body of the response is a *Service Representations List Representation* from the REST management service. The request can return the following status codes:

• 200 (OK): the request has been served successfully and the body of the response is a service list representation.
• 400 (BAD REQUEST): the provided filter expression was not valid.
• 406 (NOT ACCEPTABLE): the REST management service does not support any of the requested representations.

**Service Resource**

The service resource represents a single, distinct service in the framework. Hence, it has to be qualified by a service id. Services can only be read through the REST Management Service and therefore only support the GET request.

**GET**

The GET request retrieves a *Service Representation*. The request can return the following status codes:
137.4 Representations

137.4.1 Bundle Representation

137.4.1.1 JSON

Content-Type: application/org.osgi.bundle+json

```json
{
    "id":0,
    "lastModified":1314999275542,
    "location":"System Bundle",
    "state":32,
    "symbolicName":"org.eclipse.osgi",
    "version":"3.7.0.v20110613"
}
```

137.4.1.2 XML

Content-Type: application/org.osgi.bundle+xml

```xml
<bundle>
    <id>0</id>
    <lastModified>1314999275542</lastModified>
    <location>System Bundle</location>
    <state>32</state>
    <symbolicName>org.eclipse.osgi</symbolicName>
    <version>3.7.0.v20110613</version>
</bundle>
```

137.4.2 Bundles Representations

137.4.2.1 Bundle List Representation

137.4.2.1.1 JSON

Content-Type: application/org.osgi.bundles+json

```json
{[bundleURI, bundleURI, ..., bundleURI]
}
```

137.4.2.1.2 XML

Content-Type: application/org.osgi.bundles+xml

```xml
<bundles>
    <uri>bundleURI</uri>
    <uri>bundleURI</uri>
    ...
</bundles>
```
137.4.2.2 **Bundle Representations List Representation**

137.4.2.2.1 **JSON**

Content-Type: application/org.osgi.bundles.representations+json

[BUNDLE REPRESENTATION, BUNDLE REPRESENTATION, ... , BUNDLE REPRESENTATION]

137.4.2.2.2 **XML**

Content-Type: application/org.osgi.bundles.representations+xml

```xml
<bundles>
  BUNDLE REPRESENTATION
  BUNDLE REPRESENTATION
  ...
  BUNDLE REPRESENTATION
</bundles>
```

137.4.3 **Bundle State Representation**

137.4.3.1 **JSON**

Content-Type: application/org.osgi.bundlestate+json

```json
{
  "state":32,
  "options":1
}
```

137.4.3.2 **XML**

Content-Type: application/org.osgi.bundlestate+xml

```xml
<bundleState>
  <state>32</state>
  <options>1</options>
</bundleState>
```

137.4.4 **Bundle Header Representation**

137.4.4.1 **JSON**

Content-Type: application/org.osgi.bundleheader+json

```json
{
  key:value,
  key:value,
  ...
  key:value
}
```

137.4.4.2 **XML**

Content-Type: application/org.osgi.bundleheader+xml

```xml
<bundleHeader>
```
137.4.5 Framework Startlevel Representation

137.4.5.1 JSON
Content-Type: application/org.osgi.frameworkstartlevel+json

```json
{
  "startLevel":6,
  "initialBundleStartLevel":4
}
```

137.4.5.2 XML
Content-Type: application/org.osgi.frameworkstartlevel+xml

```xml
<frameworkStartLevel>
  <startLevel>6</startLevel>
  <initialBundleStartLevel>4</initialBundleStartLevel>
</frameworkStartLevel>
```

137.4.6 Bundle Startlevel Representation

137.4.6.1 JSON
Content-Type: application/org.osgi.bundlestartlevel+json

```json
{
  "startLevel":6,
  "activationPolicyUsed":true,
  "persistentlyStarted":false
}
```

137.4.6.2 XML
Content-Type: application/org.osgi.bundlestartlevel+xml

```xml
<bundleStartLevel>
  <startLevel>6</startLevel>
  <activationPolicyUsed>true</activationPolicyUsed>
  <persistentlyStarted>false</persistentlyStarted>
</bundleStartLevel>
```

137.4.7 Service Representation

137.4.7.1 JSON
Content-Type: application/org.osgi.service+json

```json
{
  properties:
  {
```
Representations

key: value,
key: value,
...
key: value
},
"bundle": bundleURI,
"usingBundles": [bundleURI, bundleURI, ... bundleURI]

137.4.7.2 XML

Content-Type: application/org.osgi.service+xml

<service>
  <properties>
    <entry key="key" value="value"/>
    <entry key="key" value="value"/>
    ...
    <entry key="key" value="value"/>
  </properties>
  <bundle>bundleURI</bundle>
  <usingBundles>
    <bundle>bundleURI</bundle>
    <bundle>bundleURI</bundle>
    ...
    <bundle>bundleURI</bundle>
  </usingBundles>
</service>

137.4.8 Services Representations

137.4.8.1 Service List Representation

137.4.8.1.1 JSON

Content-Type: application/org.osgi.services+json

{
  [serviceURI, serviceURI, ..., serviceURI]
}

137.4.8.1.2 XML

Content-Type: application/org.osgi.service+xml

<service>
  <properties>
    <entry key="key" value="value"/>
    <entry key="key" value="value"/>
    ...
    <entry key="key" value="value"/>
  </properties>
  <bundle>bundleURI</bundle>
  <usingBundles>
    <bundle>bundleURI</bundle>
    <bundle>bundleURI</bundle>
    ...
    <bundle>bundleURI</bundle>
137.4.8.2 Service Representations List Representation

137.4.8.2.1 JSON
Content-Type: org.osgi.services.representations+json

[SERVICE REPRESENTATION, SERVICE REPRESENTATION, ..., SERVICE REPRESENTATION]

137.4.8.2.2 XML
Content-Type: application/org.osgi.services.representations+xml

<services>
  SERVICE REPRESENTATION
  SERVICE REPRESENTATION
  ...
  SERVICE REPRESENTATION
</services>

137.4.9 Bundle Exception Representation

137.4.9.1 JSON
Content-Type: application/org.osgi.bundleexception+json

{
  "typecode": 5,
  "message": "BundleException: Bundle activation error"
}

137.4.9.2 XML
Content-Type: application/org.osgi.bundleexception+xml

<bundleexception>
  <typecode>5</typecode>
  <message>BundleException: Bundle activation error</message>
</bundleexception>

137.5 Extending the REST Management Service

This specification describes a REST-based management interface for the core framework functionality. Other services on the framework might also benefit from management access through REST. This can involve services specified by the OSGi Alliance as part of the Core Framework, Compendium, or Enterprise Specifications but also application-specific services provided by the developer. It is desirable to expose such management services as extensions of the REST Management Service.

This REST service can be implemented by using various technologies such as Java Servlets, Restlet, JAX-RS, and others. Therefore, it might not always be possible to integrate extensions at the implementation level because they might use other underlying technologies to implement their REST interface. Defining a format for delegating requests between the REST Management Service and extensions would furthermore necessarily expose implementation details and is therefore not feasible either. As a consequence, this specification only describes how to logically integrate extensions with the REST Management Service. Implementations of this specification might offer mechanisms for tighter integration for the case that extensions are developed using the same underlying technology.
The main purpose of the extension mechanism is therefore to make extensions discoverable by the client and give it a mechanism to check if a REST interface exists for a specific service. This is made possible by the ??? which contains a description and a path for every extension currently available. Services who want to contribute their extensions to the REST Management Service can do so by registering the RestApiExtension service using the Whiteboard Pattern. The extension interface is only a marker and the relevant information is exposed through the NAME, URI_PATH, and optionally SERVICE properties.

In order to be discoverable REST interface extensions to OSGi core, compendium, or enterprise services must use their canonical package name as names. User-defined extensions should use the package name of the service they provide management capabilities for.

### 137.5.1 Extension Resource

The extension resource enumerates all extensions currently registered through the Whiteboard Pattern. It is read-only and therefore only support the GET request.

#### 137.5.1.1 GET

The GET request retrieves a Extension List Representation. The request can return the following status codes:

- 200 (OK): the request has been served successfully and the body of the response is a extension list representation.
- 406 (NOT ACCEPTABLE): the REST management service does not support any of the requested representations.

### 137.5.2 Extension List Representation

#### 137.5.2.1 JSON

Content-Type: application/org.osgi.extensions+json

```
{
  [ { "name" : "org.osgi.service.event", "path" : "extensions/eventadmin", "service" : 12 }, ... ]
}
```

#### 137.5.2.2 XML

Content-Type: application/org.osgi.extensions+xml

```
<extensions>
  <extension>
    <name>org.osgi.service.event</name>
    <path>extensions/eventadmin</path>
    <service>12</service>
  </extension>
</extensions>
```

### 137.6 org.osgi.service.rest

Rest Service Package Version 1.0.

#### 137.6.1 public interface RestApiExtension

Marker interface for registering extensions to the Rest API service.
The REST service provides a RESTful interface to clients who want to manage an OSGi framework through a network connection. Any other components running on the same framework which would like to contribute their own specific REST interface and make it available and discoverable should register a marker service using the whiteboard pattern.

Integration of third-party REST interfaces with the framework REST service on the implementation level might not always be possible since it requires knowledge about the underlying implementation and an extension mechanism on that level. Specific technologies like, e.g., using servlets might support this but the REST service could as well be implemented without the use of a supporting abstraction layer and not offer extensibility.

Using the marker service, the REST service can provide a common directory page through which clients can discover the presence of a REST API extension on the managed framework and derive a URI through which they get access to it.

137.6.1.1 public static final String NAME = "org.osgi.rest.name"
this service property describes the name of the service who wants to contribute its REST API extension. Services specified in OSGi specifications must use their canonical package name as the name. Third-party services should use their package names. The type of this property is java.lang.String and the property is mandatory.

137.6.1.2 public static final String SERVICE = "org.osgi.rest.service"
this service property describes the refers to the id of the service the the REST API extension provides management capabilities for. Clients can use this information to retrieve the service properties of the service, e.g., for the case that there are multiple services of the same type on the framework and the client wants to retrieve the REST management extension for a specific service instance. The type of the property is java.lang.Long and the property is optional.

137.6.1.3 public static final String URI_PATH = "org.osgi.rest.uri.path"
this service property describes a URI to the REST extension on this local machine. It is either an absolute URI with a different port if no integration with the framework REST service is possible or a relative URL implicitly using the same port if integration is possible. In either case, the path to the extension must be absolute and must not start with "framework/" or "extensions/". The type of this property is java.lang.String and the property is mandatory.

137.7 org.osgi.service.rest.client

137.7.1 public interface RestClient
Provides a Java client API for accessing and managing a remote OSGi framework through the REST API. Implementations of this interface will usually take the URL to the remote REST Management Service instance as an argument in their constructor. Further arguments might be needed, e.g., if the cloud provider requires URL signing.

137.7.1.1 public BundleDTO getBundle(long id) throws Exception
id Addresses the bundle by its identifier.
□ Retrieve the bundle representation for a given bundle Id.

Returns A BundleDTO for the requested bundle.

Throws Exception–

137.7.1.2 public BundleDTO getBundle(String bundlePath) throws Exception
bundlePath Addresses the bundle by its URI path.
Retrieve the bundle representation for a given bundle path.

Returns  A BundleDTO for the requested bundle.

Throws   Exception–

public Map<String, Object> getBundleHeaders(long id) throws Exception

id   Addresses the bundle by its identifier.

Get the headers for a bundle given by its bundle Id.

Returns  Returns the map of headers entries.

Throws   Exception–

public Map<String, Object> getBundleHeaders(String bundlePath) throws Exception

bundlePath  Addresses the bundle by its URI path.

Get the headers for a bundle given by its URI path.

Returns  Returns the map of headers entries.

Throws   Exception–

public Collection<BundleDTO> getBundleRepresentations() throws Exception

Get the bundle representations for all bundles currently installed in the managed framework.

Returns  Returns a collection of BundleDTO objects.

Throws   Exception–

public Collection<String> getBundles() throws Exception

Get the bundles currently installed on the managed framework.

Returns  Returns a collection of the bundle URIs in the form of Strings. The URIs are relative to the REST API root URL and can be used to retrieve bundle representations.

Throws   Exception–

public BundleStartLevelDTO getBundleStartLevel(long id) throws Exception

id   Addresses the bundle by its identifier.

Get the start level for a bundle given by its bundle Id.

Returns  Returns a BundleStartLevelDTO describing the current start level of the bundle.

Throws   Exception–

public BundleStartLevelDTO getBundleStartLevel(String bundlePath) throws Exception

bundlePath  Addresses the bundle by its URI path.

Get the start level for a bundle given by its URI path.

Returns  Returns a BundleStartLevelDTO describing the current start level of the bundle.

Throws   Exception–

public int getBundleState(long id) throws Exception

id   Addresses the bundle by its identifier.

Get the state for a given bundle Id.

Returns  Returns the current bundle state as defined in (@link org.osgi.framework.Bundle).
137.7.1.10  public int getBundleState(String bundlePath) throws Exception

bundlePath  Addresses the bundle by its URI path.
   □  Get the state for a given bundle path.

Returns  Returns the current bundle state as defined in (@link org.osgi.framework.Bundle).  

Throws  Exception–

137.7.1.11  public FrameworkStartLevelDTO getFrameworkStartLevel() throws Exception

   □  Retrieves the current framework start level.

Returns  Returns the current framework start level in the form of a FrameworkStartLevelDTO.  

Throws  Exception–

137.7.1.12  public ServiceReferenceDTO getServiceReference(long id) throws Exception

id  Addresses the service by its identifier.
   □  Get the service representation for a service given by its service Id.

Returns  The service representation as ServiceReferenceDTO.  

Throws  Exception–

137.7.1.13  public ServiceReferenceDTO getServiceReference(String servicePath) throws Exception

servicePath  Addresses the service by its URI path.
   □  Get the service representation for a service given by its URI path.

Returns  The service representation as ServiceReferenceDTO.  

Throws  Exception–

137.7.1.14  public Collection<ServiceReferenceDTO> getServiceRepresentations() throws Exception

   □  Get the service representations for all services.

Returns  Returns the service representations in the form of ServiceReferenceDTO objects.  

Throws  Exception–

137.7.1.15  public Collection<ServiceReferenceDTO> getServiceRepresentations(String filter) throws Exception

filter  Passes a filter to restrict the result set.
   □  Get the service representations for all services.

Returns  Returns the service representations in the form of ServiceReferenceDTO objects.  

Throws  Exception–

137.7.1.16  public Collection<String> getServices() throws Exception

   □  Gets a collection of URI paths to all installed services.

Returns  Returns a collection of URI paths to the installed services.  

Throws  Exception–

137.7.1.17  public Collection<String> getServices(String filter) throws Exception

filter  Passes a filter to restrict the result set.
   □  Gets a collection of URI paths to all installed services.

Returns  Returns a collection of URI paths to the installed services.
137.7.1.18  public String installBundle(String url) throws Exception

url  Passes the location string to retrieve the bundle content from.

□ Install a new bundle given by an externally reachable location string, typically describing a URL.

Returns  Returns the URI path of the newly installed bundle.

137.7.1.19  public String installBundle(String location, InputStream in) throws Exception

location  Passes the location string to be used to install the new bundle.
in  Passes the input stream to a bundle.

□ Install a new bundle given by an InputStream to a bundle content.

Returns  Returns the URI path of the newly installed bundle.

137.7.1.20  public void setBundleStartLevel(long id, BundleStartLevelDTO startLevel) throws Exception

id  Addresses the bundle by its identifier.

startLevel  Pass the target start level in the form of a BundleStartLevelDTO.

□ Set the start level for a bundle given by its bundle Id.

137.7.1.21  public void setBundleStartLevel(String bundlePath, BundleStartLevelDTO startLevel) throws Exception

bundlePath  Addresses the bundle by its URI path.

startLevel  Pass the target start level in the form of a BundleStartLevelDTO.

137.7.1.22  public void setFrameworkStartLevel(FrameworkStartLevelDTO startLevel) throws Exception

startLevel  set the framework start level to this target.

□ Sets the current framework start level.

137.7.1.23  public void startBundle(long id) throws Exception

id  Addresses the bundle by its identifier.

□ Start a bundle given by its bundle Id.

137.7.1.24  public void startBundle(String bundlePath) throws Exception

bundlePath  Addresses the bundle by its URI path.

□ Start a bundle given by its URI path.

137.7.1.25  public void startBundle(long id, int options) throws Exception

id  Addresses the bundle by its identifier.

options  Passes additional options as defined in org.osgi.framework.Bundle.start(int)
Start a bundle given by its bundle Id.

Throws Exception –

137.7.1.26 public void startBundle(String bundlePath, int options) throws Exception

bundlePath Addresses the bundle by its URI path.

options Passes additional options as defined in org.osgi.framework.Bundle.start(int)

Start a bundle given by its URI path.

Throws Exception –

137.7.1.27 public void stopBundle(long id) throws Exception

id Addresses the bundle by its identifier.

Stop a bundle given by its bundle Id.

Throws Exception –

137.7.1.28 public void stopBundle(String bundlePath) throws Exception

bundlePath Addresses the bundle by its URI path.

Stop a bundle given by its URI path.

Throws Exception –

137.7.1.29 public void stopBundle(long id, int options) throws Exception

id Addresses the bundle by its identifier.

options Passes additional options as defined in org.osgi.framework.Bundle.stop(int)

Stop a bundle given by its bundle Id.

Throws Exception –

137.7.1.30 public void stopBundle(String bundlePath, int options) throws Exception

bundlePath Addresses the bundle by its URI path.

options Passes additional options as defined in org.osgi.framework.Bundle.stop(int)

Stop a bundle given by its URI path.

Throws Exception –

137.7.1.31 public void uninstallBundle(long id) throws Exception

id Addresses the bundle by its identifier.

Uninstall a bundle given by its bundle Id.

Throws Exception –

137.7.1.32 public void uninstallBundle(String bundlePath) throws Exception

bundlePath Addresses the bundle by its URI path.

Uninstall a bundle given by its URI path.

Throws Exception –

137.7.1.33 public void updateBundle(long id) throws Exception

id Addresses the bundle by its identifier.

Updates a bundle given by its bundle Id using the bundle-internal update location.
### TODO: add a way to generate the documentation from the annotations in the Javascript code.

#### 137.9 XML Schema

### TODO: needs schema

#### 137.10 Security

Like any externally visible management interface, the REST interface exposes privileged operations and hence requires access control. Since REST builds upon the HTTP(s) protocol, authentication mechanisms and encryption can be applied the same way as usually done for web servers: they can be layered below the REST protocol. E.g., confidentiality of the transmitted commands can be ensured by using HTTPS as the underlying transport. Authentication can be added by requiring, e.g., basic authentication prior to accepting a REST command. The REST interface should only be implemented by a trusted bundle. Implementations of this specification require all admin permissions and all service permissions.

#### 137.11 References

    OSGi Core, Chapter 3.2.7 Filter Syntax

    http://www.w3.org/TR/xmlschema-2/
138  Asynchronous Service Specification

Version 1.0

138.1  Introduction

OSGi Bundles collaborate using loosely coupled services registered in the OSGi service registry. This is a powerful and flexible model, and allows for the dynamic replacement of services at runtime. OSGi services are therefore a very common interaction pattern within OSGi.

As with most Java APIs and Objects, OSGi services are primarily synchronous in operation. This has several benefits; synchronous APIs are typically easier to write and to use than asynchronous ones; synchronous APIs provide immediate feedback; synchronous implementations typically have a less complex threading model.

Asynchronous APIs, however, have different advantages. Asynchronous APIs can reduce bottlenecks by encouraging more effective use of parallelism, improving the responsiveness of the application. In many cases high throughput systems can be written more simply and elegantly using asynchronous programming techniques.

The Promises Specification on page 699 provides powerful primitives for asynchronous programming, including the ability to compose flows in a functional style. There are, however, many existing services that do not use the Promise API. The purpose of the Asynchronous Service is to bridge the gap between these existing, primarily synchronous, services in the OSGi service registry, and asynchronous programming. The Asynchronous Service therefore provides a way to invoke arbitrary OSGi services asynchronously, providing results and failure notifications through the Promise API.

138.1.1  Essentials

- Async Invocation - A single method call that is to be executed without blocking the requesting thread.
- Client - Application code that wishes to invoke one or more OSGi services asynchronously.
- Async Service - The OSGi service representing the Asynchronous Services implementation. Used by the client to make one or more Async Invocations.
- Async Mediator - A mediator object created by the Async Service which represents the target service. Used by the Client to register async invocations.
- Success Callback - A callback made when an async invocation exits with a normal return value.
- Failure Callback - A callback made when an async invocation exits by throwing an exception.

138.1.2  Entities

- Async Service - A service that can create Async Mediators and run Async Invocations.
- Target Service - A service that is to be called asynchronously by the client.
- Client - The code that makes async invocations using the Async Service
- Promise - A promise, representing the result of the Async Invocation.
138.2 Usage

This section is an introduction in the usage of the Async service. It is not the formal specification, the normative part starts at Async Service on page 680. This section leaves out some of the details for clarity.

138.2.1 Synopsis

The Async service provides a mechanism for a client to *asynchronously* invoke methods on a target service. The service may be aware of the asynchronous nature of the call and actively participate in it, or be unaware and execute normally. In either case the client's thread will not block, and will continue executing its next instructions. Clients are notified of the completion of their task, and whether it was successful or not, through the use of the Promise API.

Each async invocation is registered by the client making a method call on an Async Mediator, and then started by making a call to the Async service that created the mediator. This call returns a Promise that will eventually be resolved with the return value from the async invocation.

An Async Mediator can be created by the client, either from an Object, or directly from a Service Reference. Using a service reference has the advantage that the mediator will track the underlying service. This means that if the service is unregistered before the asynchronous call begins then the Promise will resolve with a failure, rather than continuing using an invalid service object.

138.2.2 Making Async Invocations

The general pattern for a client is to obtain the Async service, and a service reference for the target service. The client then creates an Async Mediator for the target service, invokes a method on the mediator, then starts the asynchronous call. This is demonstrated in the following example:

```java
private Async asyncService;
private ServiceReference<Foo> fooRef;

@Reference
void setAsync(Async async) {
    asyncService = async;
}
```
This example demonstrates how simply clients can make asynchronous calls using the Async service. The eventual result can be obtained from the promise using one of the relevant callbacks.

One important thing to note is that whilst the call to `asyncService.call(...)` causes the async invocation to begin, the actual execution of the underlying task may be queued until a thread is available to run it. If the service has been unregistered before the execution actually begins then the promise will be resolved with a Service Exception. The type of the service exception will be `ASYNC_ERROR`.

### 138.2.3 Async invocations of void methods

The return value of the mediator method call is used to provide type information to the Async service. This, however, does not work for void methods that have no return value. In this case the client can either pass an arbitrary object to the call method, or use the zero argument version of the call method. In either case the returned promise will eventually resolve with a value of `null`. This is demonstrated in below.

```java
private Async asyncService;
private ServiceReference<Foo> fooRef;

@Reference void setAsync(Async async) {
    asyncService = async;
}

@Reference(service = Foo.class)
void setList(ServiceReference<Foo> foo) {
    fooRef = foo;
}

public synchronized void doStuff() {
    Foo mediator = asyncService.mediate(fooRef, Foo.class);
    Promise<Void> promise = asyncService
        .call(mediator.voidMethod());
    ...
}
```

### 138.2.4 Fire and Forget calls

Sometimes a client does not require any notification that an async invocation has completed. In this case the client could use one of the `call()` methods and simply discard the returned Promise object. This, however, can be wasteful of resources. The act of resolving the Promise object may be expen-
Asynchronous tasks can involve serializing the return value over a network if the remote call was asynchronous.

If the client knows that no Promise object representing the result of the asynchronous task is needed then it can signal this to the Async service. This allows the Async service to better optimize the async invocation by not providing a result.

To indicate that the client wants to make a fire-and-forget style call the client invokes the mediator as normal, but then begins the asynchronous invocation using the `execute()` method as shown below.

```java
private Async asyncService;
private ServiceReference<Foo> fooRef;

@Reference
do setAsync(Async async) {
    asyncService = async;
}

@Reference(service = Foo.class)
do setList(ServiceReference<Foo> foo) {
    fooRef = foo;
}

do public void doStuff() {
    Foo mediator = asyncService.mediate(fooRef, Foo.class);
    mediator.someMethod();
    asyncService.execute();
    ...
}
```

Note that the `execute()` method does still return a Promise. This Promise is not the same as the ones returned by `call()`, its resolution value does not provide access to the result, but instead indicates whether the fire-and-forget call could be successfully started. If there is a failure which prevents the task from being executed then this is used to fail the returned promise.

### 138.2.5 Multi Threading

By their very definition asynchronous tasks do not run inline, and typically they will not run on the same thread as the caller. This is not, however, a guarantee. A valid implementation of the Async service may have only one worker thread, which may be the thread currently running in the client code. Async invocations also have the same threading model as the Promise API. This means that callbacks may run on arbitrary threads, which may, or may not, be the same as the client thread, or the thread which executed the asynchronous work.

It is important for multi-threaded clients to note that calls to the mediator and async service must occur on the same thread. For example it is not supported to invoke a mediator using one thread, and then to begin the async invocation by calling one of the `Async.call(...)` methods on a different thread.

### 138.3 Async Service

The Async service is the primary interaction point between a client and the Async service implementation. An Async service implementation must expose a service implementing the `org.osgi.service.async.Async` interface. Clients obtain an instance of the Async service using the normal OSGi service registry mechanisms, either directly using the OSGi framework API, or using dependency injection.
The Async service is used to:

- Create async mediators
- Begin async invocations
- Obtain Promise objects representing the result of the async invocation

### 138.3.1 Using the Async service

The first action that a client wishing to make an async invocation must take is to create an async mediator using one of the `mediate` methods. Once created the client invokes the method that should be run asynchronously, supplying the arguments that should be used. This call records the invocation, but does not start the asynchronous task. The asynchronous task begins when the client invokes one of the `call` or `execute` methods on the Async service. The call methods must return a Promise representing the async invocation. The promise must resolve with the value returned by the async invocation, or fail with the failure thrown by the async invocation.

If the client attempts to begin an async invocation without first having called a method on the mediator object then the Async service must detect this usage error and throw an `IllegalStateException` to the client. This applies to all methods that begin an async invocation.

### 138.3.2 Asynchronous failures

There are a variety of reasons that async invocations may be started correctly by the client, but then fail without running the asynchronous task. In any of these cases the Promise representing the async invocation must fail with a Service Exception. This service exception must be initialised with a type of `ASYNC_ERROR`. If there is no promise representing the async invocation then there is no way to notify the client of the failure, therefore the service exception must be logged by the Async service using all available Log Service implementations.

The following list of scenarios is not exhaustive, but indicates failure scenarios that must result in a service exception with a type of async:

- If the client is using a service reference backed mediator and the client bundle's bundle context becomes invalid before looking up the target service.
- If the client is using a service reference backed mediator and the service is unregistered before making the async invocation.
- If the client is using a service reference backed mediator and the service lookup returns `null`.
- If the Async service is unable to accept new work, for example it is in the process of being shut down.
- If the type of the mediator object does not match the type of the service object to be invoked.

### 138.3.3 Thread safety and instance sharing

Implementations of the Async service must be thread safe and may be used simultaneously across multiple clients and from multiple threads within the same client. Whilst the async service is able to be used across multiple threads, if a client wishes to make an async invocation then the call to the mediator and the call to begin the async invocation must occur on the same thread. The returned Promise may then be shared between threads if required.

It is expected, although not required, that the Async service implementation will use a Service Factory to create customized implementations for each client bundle. This simplifies the tracking of the relevant client bundle context to use when performing service lookups on the client bundle's behalf. Clients should therefore not share instances of the Async service with other bundles. Instead both bundles should obtain their own instances from the service registry.
138.3.4 Service Object Lifecycle management

If the Async Service is being used to call an OSGi service object and the service reference is available then the service object should be looked up immediately before the asynchronous task begins executing. This ensures that the service is still available at the point it is eventually called. Any call to `getservice()` must have a corresponding call to `ungetService()` after the mediated method invoked has returned and, if available, the promise is resolved, but before the asynchronous task releases its thread of execution.

138.4 The Async Mediator

Async mediators are dynamically created objects that have the same type or interface as the object being mediated, and are used to record method invocations and arguments. Mediator objects are specific to an Async service implementation, and must only be used in conjunction with the async service object that they were created by.

Mediators may be created either from a `ServiceReference` or from a service object. The actions and overall result are similar for both `mediate(...)` methods, with the primary difference being that mediated objects created from a `ServiceReference` will validate whether the service object is still available immediately before the asynchronous task is executed.

138.4.1 Building the mediator object

The client passes in a `Class` indicating the type that should be mediated. If the class object represents an interface type then the generated mediator object must implement that interface. If the class object represents a Java class type then the mediator object must either be an instance of that type or extend it.

When building a mediator object the Async service has the opportunity to detect numerous problems, for example if the service reference to be mediated has been unregistered. Although fail-fast behaviour is usually preferable, in this case it would force the client to handle errors in two places; both when creating the mediator, and for the returned Promise. To simplify client usage, error cases detected when creating a mediator must not prevent the mediator from being created and must not result in an exception being thrown. The only reason that the Async service may fail to create a mediator is if the class object passed in cannot be mediated.

There are three reasons why the Async service may not be able to mediate a class type:

- The class object passed in represents a final type
- The class object passed in represents a type that has no zero-argument constructor
- The class object passed in represents a type which has one or more public final methods present in its type hierarchy (other than those declared by java.lang.Object)

If any of these constraints are violated and prevent the Async service from creating a mediator then the Async service must throw an `IllegalArgumentException`.

138.4.2 Async mediator behaviours

When invoked the Async mediator must record the method call, and its arguments, and then return rapidly (i.e. it should not perform blocking operations). The values returned by the mediator object are opaque, and the client should not attempt to interpret the returned value. The value may be null (or null-like in the case of primitives) or contain implementation specific information. If the mediated method call has a return type, specifically it is non-void, then this object must be passed to the the async service's call method when beginning the async invocation.

Async mediators should make a best-effort attempt to detect incorrect API usage from the client. If this incorrect usage is detected then the mediator object must throw an `IllegalStateException` when
invoked. An example of incorrect usage that must be detected is when a client makes multiple invo-
cations on a single mediator object from the same thread without making any calls to the Async ser-
vice.

After a usage error has been detected and an IllegalStateException has been thrown the mediator ob-
ject must be reset so that a subsequent invocation from the client thread can proceed normally.

138.4.3 Thread safety and instance sharing

Async mediators, unlike instances of the Async service, are not required to be thread safe. Clients
should not share mediator objects with other bundles, or accross threads. Also, if a client wishes to
make an async invocation then the call to the mediator and the call to async.call(...) must occur on
the same thread. The returned Promise may then be shared between threads if required.

Async mediators created from ServiceReference objects remain directly associated with the service
reference and client bundle after creation. Clients should therefore not share mediator objects with
other bundles. Instead both bundles should create their own mediators.

138.5 Fire and Forget invocations

The Async service provides call() methods for clients to use when they wish to receive results from
asynchronous tasks. Clients that do not need the result can simply discard the returned Promise ob-
ject. This, however, can be wasteful of resources. The act of resolving the Promise object may be ex-
pensive, for example it may involve serializing the return value over a network.

To address this use case the Async service must provide the execute() method, which behaves sim-
ilarly to call(), but does not provide access to the eventual result. Instead the execute() method re-
turns a Promise that indicates whether the fire-and-forget call is able to be be successfully started.

The returned Promise must be resolved with null if the asynchronous task begins executing success-
fully. There is no happens-before relationship required, meaning that if the Promise resolves suc-
cessfully then the task may, or may not, have started or finished. The primary usage of the Promise
is actually to detect failures. If the fire and forget task cannot be executed for some reason, for exam-
ple the backing service has been unregistered, then the returned promise must be failed appropri-
ately using the same rules as defined in Asynchronous failures on page 681. If the returned Promise
is failed then the fire-and-forget task has not executed and will not execute in the future.

138.6 Delegating to asynchronous implementations

Some service APIs are already asynchronous in operation, and others are partly asynchronous, in
that some methods run asynchronously and others do not. There are also services which have a syn-
chronous API, but could run asynchronously because they are a proxy to another service. A good
example of this kind of service is a remote service. Remote services are local views of a remote end-
point, and depending upon the implementation of the endpoint it may be possible to make the re-
mote call asynchronously, optimizing the thread usage of any local asynchronous call.

Services that already have some level of asynchronous support may advertise this to clients and to
the Async service by implementing org.osgi.service.async.delegate.AsyncDelegate. This interface
can be used by the Async service implementation, or by the client directly, to make an asynchro-
nous call on the service.

Because the Async Delegate behaviour is transparently handled by the async service, clients of the
async service do not need to know whether the target service implements Async Delegate or not,
their usage pattern can remain unchanged.
When making an async invocation the async service must check to see whether the target service implements Async Delegate. If the target service does implement AsyncDelegate then the async service must attempt to delegate the asynchronous call. The exact delegation pattern depends on whether a Promise is required.

138.6.1 Obtaining a Promise from an Async Delegate

If a promise is needed by the client then the async service must attempt to delegate to the async method. The delegation proceeds as follows:

- If the call to the Async Delegate returns a Promise, then the Promise returned by the async service must be resolved with that Promise.
- If the call to the Async Delegate throws an exception then this must be used to fail the promise returned by the Async service.
- If the Async Delegate is unable to optimise the call then it may return a null Promise from the async method. In this case the Async service must continue processing the async invocation, treating the service as a normal object.

138.6.2 Delegating fire and forget calls to an Async Delegate

If no promise is needed by the client then the async service must attempt to delegate to the execute method. This gives the Async Delegate implementation the opportunity to further optimise its processing. The delegation proceeds as follows:

- If the call to the Async Delegate returns true then the async service should return a successfully resolved Promise to the client.
- If the call to the Async Delegate throws an exception then this must be logged to any available Log Services, and must be used to fail the Promise returned to the client.
- If the Async Delegate is unable to optimise the call then it may return false from the execute method. In this case the Async service must continue processing the async invocation, treating the service as a normal object.

138.6.3 Lifecycle for Async Delegate Services

If an Async Delegate service implementation accepts an asynchronous task, either via a call to execute() or async(), then it is responsible for continuing to process the work until completion. This means that if the service is unregistered for some reason then the task must be properly cleaned up and succeed or fail as appropriate.

If the Async services implementation used a service reference to obtain the Async Delegate service object then it must release the service object after the task has been accepted. This means that if the Async Delegate service object is provided by a service factory then the service object should take extra care not to destroy its internal state when released. The service object must remain valid until all executing asynchronous tasks are either completed or failed.

In the case where an AsyncDelegate service rejects an asynchronous task with a return of false or null, the Async service implementation will take over the execution of the task. In this case the Async service should not release the service object until the asynchronous task is completed. If the asynchronous task is rejected with an Exception then the service must be released immediately.

138.7 Security

Asynchronous Services implementations must be careful to avoid elevating the privileges of client bundles when calling services asynchronously, and also to avoid restricting the privileges of clients that are permitted to make a call. This means that the implementation must:
• Be granted AllPermission. As the Async service will always be on the stack when invoking a service object asynchronously it must be granted AllPermission so that it does not interfere with security any checks made by the service object.

• Establish the caller's AccessControlContext in a worker thread before starting to call the service object. This prevents a bundle from being able to call a service asynchronously that it would not normally be able to call. The AccessControlContext must be collected during any call to call() or execute().

• Use a doPrivileged block when mediating a concrete type. A no-args constructor in a concrete type may perform actions that the client may not have permission to perform. This should not prevent the client from mediating the object, as the client is not directly performing these actions.

• If the mediator object was created using a service reference, then the Async services implementation must use the client's bundle context when retrieving the target service. If the service lookup occurs on a worker thread then the lookup must use the AccessControlContext collected during the call to call() or execute(). This prevents the client bundle from being able to make calls on a service object that they do not have permission to obtain, and ensures that an appropriately customised object is returned if the service is implemented using a service factory.

Further security considerations can be addressed using normal OSGi security rules. For example access to the Async service can be controlled using ServicePermission[Async, GET].

138.8 org.osgi.service.async

Asynchronous Services 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.async; version="[1.0,2.0)"`

Example import for providers implementing the API in this package:

`Import-Package: org.osgi.service.async; version="[1.0,1.1)"

138.8.1 public interface Async

The Asynchronous Execution Service. This can be used to make asynchronous invocations on OSGi services and objects through the use of a mediator object.

Typical usage:

```java
Async async = ctx.getService(asyncRef);

ServiceReference<MyService> ref = ctx.getServiceReference(MyService.class);

MyService asyncMediator = async.mediate(ref);

Promise<BigInteger> result = async.call(asyncMediator.getSumOverAllValues());
```

The Promise API allows callbacks to be made when asynchronous tasks complete, and can be used to chain Promises.

Multiple asynchronous tasks can be started concurrently, and will run in parallel if the Async service has threads available.
138.8.1.1 public Promise<R> call(R r)

Type Arguments

<r>

r the return value of the mediated call, used for type information

This method launches the last method call registered by a mediated object as an asynchronous task. The result of the task can be obtained using the returned promise.

Typically the parameter for this method will be supplied inline like this:

I i = async.mediate(s, I.class);
Promise<String> p = async.call(i.foo());

Returns a Promise which can be used to retrieve the result of the asynchronous execution

138.8.1.2 public Promise<?> call()

This method launches the last method call registered by a mediated object as an asynchronous task. The result of the task can be obtained using the returned promise.

Generally it is preferable to use call(Object) like this:

I i = async.mediate(s, I.class);
Promise<String> p = async.call(i.foo());

However this pattern does not work for void methods. Void methods can therefore be handled like this:

I i = async.mediate(s, I.class);
i.voidMethod();
Promise<?> p = async.call();

Returns a Promise which can be used to retrieve the result of the asynchronous execution

138.8.1.3 public Promise<Void> execute()

This method should be used by clients in preference to call() and call(Object) when no callbacks, or other features of Promise, are needed.

The advantage of the execute() method is that it allows for greater optimisation of the underlying asynchronous execution. Clients are therefore likely to see better performance when using this method compared to using call() and discarding the return value.

This method launches the last method call registered by a mediated object as an asynchronous task. The task runs as a “fire and forget” process, and there will be no notification of its eventual success or failure. The Promise returned by this method is different from the Promise returned by call(), in that the returned Promise will resolve when the fire and forget task is successfully started, or fail if the task cannot be started. Note that there is no happens-before relationship and the returned Promise may resolve before or after the fire-and-forget task starts, or completes.

Typically this method is used like call():

I i = async.mediate(s, I.class);
i.someMethod()
Promise<?> p = async.execute();

Returns a promise representing whether the fire and forget task was able to start

138.8.1.4 public T mediate(T target, Class<T> iface)

Type Arguments

<T>
target  The service object to mediate
iface  The type that the mediated object should provide

Create a mediator for the given object. The mediator is a generated object that registers the method calls made against it. The registered method calls can then be run asynchronously using either the call(Object) or call() method.

The values returned by method calls made on a mediated object should be ignored.

Normal usage:

```java
I i = async.mediate(s, I.class);
Promise<String> p = async.call(i.foo());
```

Returns  A mediator for the service object
Throws  IllegalArgumentException – if the type represented by iface cannot be mediated

138.8.1.5
```
public T mediate(ServiceReference<T> target, Class<T> iface)
```

Type Arguments  <T>

target  The service reference to mediate
iface  The type that the mediated object should provide

Create a mediator for the given service. The mediator is a generated object that registers the method calls made against it. The registered method calls can then be run asynchronously using either the call(Object) or call() method.

The values returned by method calls made on a mediated object should be ignored.

This method differs from mediate(Object, Class) in that it can track the availability of the backing service. This is recommended as the preferred option for mediating OSGi services as asynchronous tasks may not start executing until some time after they are requested. Tracking the validity of the ServiceReference for the service ensures that these tasks do not proceed with an invalid object.

Normal usage:

```java
I i = async.mediate(s, I.class);
Promise<String> p = async.call(i.foo());
```

Returns  A mediator for the service object
Throws  IllegalArgumentException – if the type represented by iface cannot be mediated

138.9  org.osgi.service.async.delegate

Asynchronous Services Delegation 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 contains only interfaces that are implemented by consumers.

Example import for consumers using the API in this package:
```
Import-Package: org.osgi.service.async; version="[1.0,2.0)"
```

138.9.1  public interface AsyncDelegate

This interface is used by services to allow them to optimize Asynchronous calls where they are capable of executing more efficiently. This may mean that the service has access to its own thread pool, or that it can delegate work to a remote node, or act in some other way to reduce the load on the Asynchronous Services implementation when making an asynchronous call.
138.9.1.1 public Promise<?> async(Method m, Object[] args) throws Exception

- **m** the method that should be asynchronously executed
- **args** the arguments that should be used to invoke the method

This method can be used by clients, or the Async service, to optimize Asynchronous execution of methods. When called, the AsyncDelegate should execute the supplied method using the supplied arguments asynchronously, returning a promise that can be used to access the result. If the method cannot be executed asynchronously by the delegate then it should return null.

**Returns** A promise representing the asynchronous result, or null if this method cannot be asynchronously invoked by the AsyncDelegate.

**Throws** Exception – An exception should be thrown only if there was a serious error that prevented the asynchronous call from starting, for example the supplied method does not exist on this object. Exceptions should not be thrown to indicate that the call does not support asynchronous invocation, instead the AsyncDelegate should return null. Exceptions should also not be thrown to indicate a failure from the execution of the underlying method, this should be handled by failing the returned promise.

138.9.1.2 public boolean execute(Method m, Object[] args) throws Exception

- **m** the method that should be asynchronously executed
- **args** the arguments that should be used to invoke the method

This method can be used by clients, or the Async service, to optimize Asynchronous execution of methods. When called, the AsyncDelegate should execute the supplied method using the supplied arguments asynchronously. This method differs from async(Method, Object[]) in that it does not return a promise. This method therefore allows the implementation to perform more aggressive optimisations because the end result of the invocation does not need to be returned to the client. If the method cannot be executed asynchronously by the delegate then it should return false.

**Returns** true if the asynchronous execution request has been accepted, or false if this method cannot be asynchronously invoked by the AsyncDelegate.

**Throws** Exception – An exception should be thrown only if there was an serious error that prevented the asynchronous call from starting, for example the supplied method does not exist on this object. Exceptions should not be thrown to indicate that the call does not support asynchronous invocation, instead the AsyncDelegate should return false. Exceptions should also not be thrown to indicate a failure from the execution of the underlying method.

138.10 References

[1] OSGi Core Release 6
http://www.osgi.org/Specifications/HomePage
702 XML Parser Service Specification

Version 1.0

702.1 Introduction

The Extensible Markup Language (XML) has become a popular method of describing data. As more bundles use XML to describe their data, a common XML Parser becomes necessary in an embedded environment in order to reduce the need for space. Not all XML Parsers are equivalent in function, however, and not all bundles have the same requirements on an XML parser.

This problem was addressed in the Java API for XML Processing, see [4] JAXP for Java 2 Standard Edition and Enterprise Edition. This specification addresses how the classes defined in JAXP can be used in an OSGi framework. It defines how:

- Implementations of XML parsers can become available to other bundles
- Bundles can find a suitable parser
- A standard parser in a JAR can be transformed to a bundle

702.1.1 Essentials

- **Standards** - Leverage existing standards in Java based XML parsing: JAXP, SAX and DOM
- **Unmodified JAXP code** - Run unmodified JAXP code
- **Simple** - It should be easy to provide a SAX or DOM parser as well as easy to find a matching parser
- **Multiple** - It should be possible to have multiple implementations of parsers available
- **Extendable** - It is likely that parsers will be extended in the future with more functionality

702.1.2 Entities

- **XMLParserActivator** - A utility class that registers a parser factory from declarative information in the Manifest file.
- **SAXParserFactory** - A class that can create an instance of a SAXParser class.
- **DocumentBuilderFactory** - A class that can create an instance of a DocumentBuilder class.
- **SAXParser** - A parser, instantiated by a SaxParserFactory object, that parses according to the SAX specifications.
- **DocumentBuilder** - A parser, instantiated by a DocumentBuilderFactory, that parses according to the DOM specifications.
702.1.3 Operations

A bundle containing a SAX or DOM parser is started. This bundle registers a SAXParserFactory and/or a DocumentBuilderFactory service object with the Framework. Service registration properties describe the features of the parsers to other bundles. A bundle that needs an XML parser will get a SAXParserFactory or DocumentBuilderFactory service object from the Framework service registry. This object is then used to instantiate the requested parsers according to their specifications.

702.2 JAXP

XML has become very popular in the last few years because it allows the interchange of complex information between different parties. Though only a single XML standard exists, there are multiple APIs to XML parsers, primarily of two types:

- The Simple API for XML (SAX1 and SAX2)
- Based on the Document Object Model (DOM 1 and 2)

Both standards, however, define an abstract API that can be implemented by different vendors. A given XML Parser implementation may support either or both of these parser types by implementing the org.w3c.dom and/or org.xml.sax packages. In addition, parsers have characteristics such as whether they are validating or non-validating parsers and whether or not they are namespace aware.

An application which uses a specific XML Parser must code to that specific parser and become coupled to that specific implementation. If the parser has implemented [4] JAXP, however, the application developer can code against SAX or DOM and let the runtime environment decide which parser implementation is used.

JAXP uses the concept of a factory. A factory object is an object that abstracts the creation of another object. JAXP defines a DocumentBuilderFactory and a SAXParserFactory class for this purpose.
JAXP is implemented in the `javax.xml.parsers` package and provides an abstraction layer between an application and a specific XML Parser implementation. Using JAXP, applications can choose to use any JAXP compliant parser without changing any code, simply by changing a System property which specifies the SAX- and DOM factory class names.

In JAXP, the default factory is obtained with a static method in the `SAXParserFactory` or `DocumentBuilderFactory` class. This method will inspect the associated System property and create a new instance of that class.

### 702.3 XML Parser service

The current specification of JAXP has the limitation that only one of each type of parser factories can be registered. This specification specifies how multiple `SAXParserFactory` objects and `DocumentBuilderFactory` objects can be made available to bundles simultaneously.

Providers of parsers should register a JAXP factory object with the OSGi service registry under the factory class name. Service properties are used to describe whether the parser:

- Is validating
- Is name-space aware
- Has additional features

With this functionality, bundles can query the OSGi service registry for parsers supporting the specific functionality that they require.

### 702.4 Properties

Parsers must be registered with a number of properties that qualify the service. In this specification, the following properties are specified:

- **PARSER_NAMESPACEAWARE** - The registered parser is aware of name-spaces. Name-spaces allow an XML document to consist of independently developed DTDs. In an XML document, they are recognized by the `xmlns` attribute and names prefixed with an abbreviated name-space identifier, like: `xmlns:xs ...`. The type is a `Boolean` object that must be `true` when the parser supports name-spaces. All other values, or the absence of the property, indicate that the parser does not implement name-spaces.

- **PARSER_VALIDATING** - The registered parser can read the DTD and can validate the XML accordingly. The type is a `Boolean` object that must be `true` when the parser is validating. All other values, or the absence of the property, indicate that the parser does not validate.

### 702.5 Getting a Parser Factory

Getting a parser factory requires a bundle to get the appropriate factory from the service registry. In a simple case in which a non-validating, non-name-space aware parser would suffice, it is best to use `getServiceReference(String)`.

```java
DocumentBuilder getParser(BundleContext context)
    throws Exception {
        ServiceReference ref = context.getServiceReference(
            DocumentBuilderFactory.class.getName() );
        if ( ref == null )
            return null;
```
DocumentBuilderFactory factory = 
  (DocumentBuilderFactory) context.getService(ref);
return factory.newDocumentBuilder();
}

In a more demanding case, the filtered version allows the bundle to select a parser that is validating and name-space aware:

SAXParser getParser(BundleContext context)
  throws Exception {
    ServiceReference refs[] = context.getServiceReferences(
      SAXParserFactory.class.getName(),
      "(&amp;parser.namespaceAware=true)"
      + "(&amp;parser.validating=true))");
    if ( refs == null )
      return null;
    SAXParserFactory factory =
      (SAXParserFactory) context.getService(refs[0]);
    return factory.newSAXParser();
  }

702.6 Adapting a JAXP Parser to OSGi

If an XML Parser supports JAXP, then it can be converted to an OSGi aware bundle by adding a BundleActivator class which registers an XML Parser Service. The utility org.osgi.util.xml.XMLParserActivator class provides this function and can be added (copied, not referenced) to any XML Parser bundle, or it can be extended and customized if desired.

702.6.1 JAR Based Services

Its functionality is based on the definition of the [5] JAR File specification, services directory. This specification defines a concept for service providers. A JAR file can contain an implementation of an abstractly defined service. The class (or classes) implementing the service are designated from a file in the META-INF/services directory. The name of this file is the same as the abstract service class.

The content of the UTF-8 encoded file is a list of class names separated by new lines. White space is ignored and the number sign (#) is the comment character.

JAXP uses this service provider mechanism. It is therefore likely that vendors will place these service files in the META-INF/services directory.

702.6.2 XMLParserActivator

To support this mechanism, the XML Parser service provides a utility class that should be normally delivered with the OSGi framework implementation. This class is a Bundle Activator and must start when the bundle is started. This class is copied into the parser bundle, and not imported.

The start method of the utility BundleActivator class will look in the META-INF/services service provider directory for the files javax.xml.parsers.SAXParserFactory (SAXFACTORYNAME) or javax.xml.parsers.DocumentBuilderFactory (DOMFACTORYNAME). The full path name is specified in the constants SAXCLASSFILE and DOMCLASSFILE respectively.

If either of these files exist, the utility BundleActivator class will parse the contents according to the specification. A service provider file can contain multiple class names. Each name is read and a new instance is created. The following example shows the possible content of such a file:

# ACME example SAXParserFactory file
Both the `javax.xml.parsers.SAXParserFactory` and the `javax.xml.parsers.DocumentBuilderFactory` provide methods that describe the features of the parsers they can create. The `XMLParserActivator` activator will use these methods to set the values of the properties, as defined in `Properties` on page 691, that describe the instances.

### 702.6.3 Adapting an Existing JAXP Compatible Parser

To incorporate this bundle activator into a XML Parser Bundle, do the following:

- If SAX parsing is supported, create a `/META-INF/services/javax.xml.parsers.SAXParserFactory` resource file containing the class names of the `SAXParserFactory` classes.
- If DOM parsing is supported, create a `/META-INF/services/javax.xml.parsers.DocumentBuilderFactory` file containing the fully qualified class names of the `DocumentBuilderFactory` classes.
- Create manifest file which imports the packages `org.w3c.dom`, `org.xml.sax`, and `javax.xml.parsers`.
- Add a Bundle-Activator header to the manifest pointing to the `XMLParserActivator`, the sub-class that was created, or a fully custom one.
- If the parsers support attributes, properties, or features that should be registered as properties so they can be searched, extend the `XMLParserActivator` class and override `setSAXProperties` and `setDOMProperties`.
- Ensure that custom properties are put into the `Hashtable` object. JAXP does not provide a way for `XMLParserActivator` to query the parser to find out what properties were added.
- Bundles that extend the `XMLParserActivator` class must call the original methods via super to correctly initialize the XML Parser Service properties.
- Compile this class into the bundle.
- Install the new XML Parser Service bundle.
- Ensure that the `org.osgi.util.xml.XMLParserActivator` class is contained in the bundle.

### 702.7 Usage of JAXP

A single bundle should export the JAXP, SAX, and DOM APIs. The version of contained packages must be appropriately labeled. JAXP 1.1 or later is required which references SAX 2 and DOM 2. See JAXP for the exact version dependencies.

This specification is related to related packages as defined in the JAXP 1.1 document. The following table contains the expected minimum versions.

<table>
<thead>
<tr>
<th>Package</th>
<th>Minimum Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.xml.parsers</td>
<td>1.1</td>
</tr>
<tr>
<td>org.xml.sax</td>
<td>2.0</td>
</tr>
<tr>
<td>org.xml.sax.helpers</td>
<td>2.0</td>
</tr>
<tr>
<td>org.xsmil.sax.ext</td>
<td>1.0</td>
</tr>
<tr>
<td>org.w3c.dom</td>
<td>2.0</td>
</tr>
</tbody>
</table>

The Xerces project from the Apache group, [6] Xerces 2 Java Parser, contains a number of libraries that implement the necessary APIs. These libraries can be wrapped in a bundle to provide the relevant packages.
702.8 Security

A centralized XML parser is likely to see sensitive information from other bundles. Provisioning an XML parser should therefore be limited to trusted bundles. This security can be achieved by providing ServicePermission[javax.xml.parsers.DocumentBuilderFactory|javax.xml.parsers.SAXFactory,REGISTER] to only trusted bundles.

Using an XML parser is a common function, and ServicePermission[javax.xml.parsers.DOMParserFactory|javax.xml.parsers.SAXFactory, GET] should not be restricted.

The XML parser bundle will need FilePermission[<<ALL FILES>>,READ] for parsing of files because it is not known beforehand where those files will be located. This requirement further implies that the XML parser is a system bundle that must be fully trusted.

702.9 org.osgi.util.xml

XML Parser Package Version 1.0.

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

Example import for consumers using the API in this package:

Import-Package: org.osgi.util.xml; version="[1.0,2.0)"

702.9.1 public class XMLParserActivator
implements BundleActivator, ServiceFactory

A BundleActivator class that allows any JAXP compliant XML Parser to register itself as an OSGi parser service. Multiple JAXP compliant parsers can concurrently register by using this BundleActivator class. Bundles who wish to use an XML parser can then use the framework's service registry to locate available XML Parsers with the desired characteristics such as validating and namespace-aware.

The services that this bundle activator enables a bundle to provide are:

- javax.xml.parsers.SAXParserFactory(SAXFACTORYNAME)
- javax.xml.parsers.DocumentBuilderFactory(DOMFACTORYNAME)

The algorithm to find the implementations of the abstract parsers is derived from the JAR file specifications, specifically the Services API.

An XMLParserActivator assumes that it can find the class file names of the factory classes in the following files:

- /META-INF/services/javax.xml.parsers.SAXParserFactory is a file contained in a jar available to the runtime which contains the implementation class name(s) of the SAXParserFactory.
- /META-INF/services/javax.xml.parsers.DocumentBuilderFactory is a file contained in a jar available to the runtime which contains the implementation class name(s) of the DocumentBuilderFactory

If either of the files does not exist, XMLParserActivator assumes that the parser does not support that parser type.

XMLParserActivator attempts to instantiate both the SAXParserFactory and the DocumentBuilderFactory. It registers each factory with the framework along with service properties.
Individual parser implementations may have additional features, properties, or attributes which could be used to select a parser with a filter. These can be added by extending this class and overriding the setSAXProperties and setDOMProperties methods.

### Concurrency
Thread-safe

#### 702.9.1.1
```java
public static final String DOMCLASSFILE = "/META-INF/services/javax.xml.parsers.DocumentBuilderFactory"
```
Fully qualified path name of DOM Parser Factory Class Name file.

#### 702.9.1.2
```java
public static final String DOMFACTORYNAME = "javax.xml.parsers.DocumentBuilderFactory"
```
Filename containing the DOM Parser Factory Class name. Also used as the basis for the SERVICE_PID registration property.

#### 702.9.1.3
```java
public static final String PARSER_NAMESPACEAWARE = "parser.namespaceAware"
```
Service property specifying if factory is configured to support namespace aware parsers. The value is of type Boolean.

#### 702.9.1.4
```java
public static final String PARSER_VALIDATING = "parser.validating"
```
Service property specifying if factory is configured to support validating parsers. The value is of type Boolean.

#### 702.9.1.5
```java
public static final String SAXCLASSFILE = "/META-INF/services/javax.xml.parsers.SAXParserFactory"
```
Fully qualified path name of SAX Parser Factory Class Name file.

#### 702.9.1.6
```java
public static final String SAXFACTORYNAME = "javax.xml.parsers.SAXParserFactory"
```
Filename containing the SAX Parser Factory Class name. Also used as the basis for the SERVICE_PID registration property.

#### 702.9.1.7
```java
public XmlParserActivator()
```

#### 702.9.1.8
```java
public Object getService(Bundle bundle, ServiceRegistration registration)
```
- **bundle** The bundle using the service.
- **registration** The ServiceRegistration object for the service.

□ Creates a new XML Parser Factory object.

□ A unique XML Parser Factory object is returned for each call to this method.

□ The returned XML Parser Factory object will be configured for validating and namespace aware support as specified in the service properties of the specified ServiceRegistration object. This method can be overridden to configure additional features in the returned XML Parser Factory object.

□ Returns A new, configured XML Parser Factory object or null if a configuration error was encountered

#### 702.9.1.9
```java
public void setDOMProperties(DocumentBuilderFactory factory, Hashtable props)
```
- **factory** - the DocumentBuilderFactory object
- **props** - Hashtable of service properties.

□ Set the customizable DOM Parser Service Properties.
This method attempts to instantiate a validating parser and a namespace aware parser to determine if the parser can support those features. The appropriate properties are then set in the specified props object.

This method can be overridden to add additional DOM2 features and properties. If you want to be able to filter searches of the OSGi service registry, this method must put a key, value pair into the properties object for each feature or property. For example, `properties.put("http://www.acme.com/features/foo", Boolean.TRUE);`

702.9.1.10  
```java
public void setSAXProperties(SAXParserFactory factory,Hashtable properties)
```

- `factory` - the SAXParserFactory object
- `properties` - the properties object for the service

Set the customizable SAX Parser Service Properties.

This method attempts to instantiate a validating parser and a namespace aware parser to determine if the parser can support those features. The appropriate properties are then set in the specified properties object.

This method can be overridden to add additional SAX2 features and properties. If you want to be able to filter searches of the OSGi service registry, this method must put a key, value pair into the properties object for each feature or property. For example, `properties.put("http://www.acme.com/features/foo", Boolean.TRUE);`

702.9.1.11  
```java
public void start(BundleContext context) throws Exception
```

- `context` - The execution context of the bundle being started.

Called when this bundle is started so the Framework can perform the bundle-specific activities necessary to start this bundle. This method can be used to register services or to allocate any resources that this bundle needs.

This method must complete and return to its caller in a timely manner.

This method attempts to register a SAX and DOM parser with the Framework’s service registry.

**Throws**  
`Exception` – If this method throws an exception, this bundle is marked as stopped and the Framework will remove this bundle’s listeners, unregister all services registered by this bundle, and release all services used by this bundle.

702.9.1.12  
```java
public void stop(BundleContext context) throws Exception
```

- `context` - The execution context of the bundle being stopped.

This method has nothing to do as all active service registrations will automatically get unregistered when the bundle stops.

**Throws**  
`Exception` – If this method throws an exception, the bundle is still marked as stopped, and the Framework will remove the bundle’s listeners, unregister all services registered by the bundle, and release all services used by the bundle.

702.9.1.13  
```java
public void ungetService(Bundle bundle,ServiceRegistration registration,Object service)
```

- `bundle` - The bundle releasing the service.
- `registration` - The ServiceRegistration object for the service.
- `service` - The XML Parser Factory object returned by a previous call to the getService method.

Releases a XML Parser Factory object.
702.10 References

[1] XML
http://www.w3.org/XML

[2] SAX
http://www.saxproject.org/

[3] DOM Java Language Binding
http://www.w3.org/TR/REC-DOM-Level-1/java-language-binding.html

[4] JAXP
http://jaxp.java.net/

[5] JAR File specification, services directory
http://download.oracle.com/javase/1.4.2/docs/guide/jar/jar.html

[6] Xerces 2 Java Parser
http://xerces.apache.org/xerces2-j/
Promises Specification

Version 1.0

Introduction

One of the fundamental pieces of an asynchronous programming model is the mechanism by which clients retrieve the result of the asynchronous task. Since Java 5, there has been a java.util.concurrent.Future interface available in the Java class libraries, which means that it is the de facto API in Java for handling the result of an asynchronous task. Futures have some limitations however in that they have no mechanism for registering callbacks. Java 8 introduces the class java.util.concurrent.CompletableFuture which addresses this but the requirement of Java 8 is unsuitable for many OSGi users at this time.

This specification defines a Promises API which can be used on many versions of Java including Java 5 and Java ME CDC/Foundation. The Promises API defined by this specification is independent of all other OSGi specifications including the OSGi Framework and thus can be easily used outside of the OSGi environment.

A Promise object holds the result of a potentially asynchronous task. The receiver of a Promise object can register callbacks on the Promise to be notified when the result is available or can block on the result becoming available. Promises can be chained together in powerful ways to handle asynchronous work flows and recovery.

Promises capture the effects of latency and errors by making these explicit in the API signatures. Latency is represented by callbacks which will eventually be called. Errors are represented by the failure member. In essence, this is what sets Promises apart from things such as RPC calls where such effects are not explicitly captured but rather attempted to be transparently handled.

Essentials

- **Common concepts** - The API is inspired by the Promises work in JavaScript and uses the same basic concepts. See [2] JavaScript Promises.
- **Independent** - The design is independent of all other OSGi specifications and can be used outside of an OSGi environment.
- **Asynchronous** - The design supports asynchronous tasks.
- **Small** - The API and implementation are very compact.
- **Complete** - The design provides a very complete set of operations for Promise which are primitives that can be used to address most use cases.
- **Resolution** - A Promise can be resolved successfully with a value or unsuccessfully with an exception.
- **Generify** - Generics are used to promote type safety.

Entities

- **Promise** - A Promise object holds the eventual result of a potentially asynchronous task.
- **Callback** - The receiver of a Promise can register callbacks on the Promise to be notified when the task is completed.
Deferred

A Deferred object represents the potentially asynchronous task and is used to resolve the Promise.

Figure 705.1 Class diagram of org.osgi.util.promise

Deferred

Promises

Promise

Failure

Success

Runnable

705.2 Promise

A Promise object holds the eventual result of a potentially asynchronous task. A Promise is either unresolved or resolved. An unresolved Promise does not have the result of the associated task available while a resolved Promise has the result of the associated task available. The `isDone()` method must return true if the Promise is resolved and false if the Promise is unresolved. A Promise must only be resolved once.

A resolved Promise can be either resolved with a value, which means the associated task completed successfully and supplied a result, or resolved with a failure, which means the associated task completed unsuccessfully and supplied an exception. The `getFailure()` method can be called to determine if the resolved Promise completed successfully with a value or unsuccessfully with a failure. If the `getFailure()` method returns a `Throwable`, the Promise resolved unsuccessfully with a failure. If the `getFailure()` method returns null, the Promise resolved successfully with a value that can be obtained from `getValue()`.

If the Promise is unresolved, then calling `getFailure()` or `getValue()` must block until the Promise is resolved. In general, these two methods should not be used outside of a callback. Use callbacks to be notified when the Promise is resolved. See Callbacks on page 701.

705.3 Deferred

Promise is an interface which can allow for many Promise implementations. This API contains the Deferred class which provides a standard Promise implementation. A Deferred object can be created with the Deferred() constructor and the Promise associated with the new Deferred object can be obtained using `getPromise()`. This Promise can then be supplied to other parties who can use it to be notified of and obtain the eventual result.

```java
public Promise<String> getTimeConsumingAnswer() {
    Deferred<String> deferred = new Deferred<String>();
    asynchronously(() -> doTask(deferred));
    return deferred.getPromise();
}
```

The creator of the Deferred object can then later resolve the Promise successfully by calling `resolve(T)` or unsuccessfully by calling `fail(Throwable)`.

```java
private void doTask(Deferred<String> deferred) {
    try {
        String answer;
```
The creator of the Deferred object can also resolve the Promise using the eventual result of another Promise using `resolveWith(Promise)`.

```java
private void doTask(Deferred<String> deferred) {
    try {
        Promise<String> answer = someMethodReturningAPromise();
        deferred.resolveWith(answer); // resolve with another Promise
    } catch (Exception e) {
        deferred.fail(e); // unsuccessfully resolve with exception
    }
}
```

If `resolve(T)` or `fail(Throwable)` is called when the Promise associated with the Deferred is already resolved, then an Illegal State Exception must be thrown.

### 705.4 Callbacks

To be notified when a Promise has been resolved, callbacks are used. The Promise API provides two forms of callbacks: the basic `Runnable` callback and the more specialized `Success` and `Failure` callbacks.

A callback may be called on a different thread than the thread which registered the callback. So the callback must be thread safe but can rely upon that the registration of the callback happens-before the callback is called.

Resolving a Promise happens-before any registered callback is called. That is, for the resolved Promise, in a registered callback `isDone()` must return true and `getValue()` and `getFailure()` must not block.

Callbacks may be registered at any time including before and after a Promise has been resolved. If a callback is registered before the Promise is resolved, it will be called later when the Promise is resolved. If a callback is registered on an already resolved Promise, it will be called right away.

### 705.4.1 Runnable

The `onResolve(Runnable)` method is used to register a Runnable with the Promise which must be called when the Promise is resolved either successfully with a value or unsuccessfully with a failure. The resolved Promise is not passed to the Runnable, so if the Runnable implementation needs access to the resolved Promise, it must take care to ensure it has access.

```java
final Promise<String> answer = getTimeConsumingAnswer();
answer.onResolve(new Runnable() {
    public void run() {
        doSomethingWithAnswer(answer);
    }
});
```

The `onResolve(Runnable)` method returns the Promise object upon which it is called.
Success and Failure

The `then(Success)` and `then(Success, Failure)` methods can be used to register the more specialized `Success` and `Failure` callbacks. The `Success` callback is only called if the Promise is successfully resolved with a value. The `Failure` is only called if the Promise is unsuccessfully resolved with a failure.

```java
Promise<String> answer = getTimeConsumingAnswer();
answer.then(p -> processResult(p.getValue()), p -> handleFailure(p.getFailure()));
```

The `then` methods return a new Promise which can be used to chain Promises together.

Chaining Promises

The `then(Success)` and `then(Success, Failure)` methods also provide a means to chain Promises together. The `then` methods return a new Promise which is chained to the original Promise upon which the `then` method was called. The returned Promise must be resolved when the original Promise is resolved after the specified Success or Failure callback is executed. The result of the executed callback must be used to resolve the returned Promise. A sequence of calls to the `then` methods can be used to create a chain of promises which are resolved in sequence.

If the original Promise is successfully resolved, the Success callback is executed and the Promise returned by the Success callback is used to resolve the Promise returned from the `then` method. If the original Promise is resolved with a failure, the Failure callback is executed and the Promise returned from the `then` method is resolved with a failure.

In the following example, a Promise which will supply the name of the file to download is chained to a Promise which will return a mirror URL to use to download the file which is then further chained to a Promise which will return an Input Stream from which to read the download file.

```java
Promise<String> name = getDownloadName();
Promise<URL> mirror = name.then(p -> getMirror(p.getValue()));
Promise<InputStream> in = mirror.then(p -> p.getValue().openStream());
```

Since we probably do not need the intermediate Promises, we can collapse the chain into a single statement.

```java
Promise<InputStream> in = getDownloadName().then(p -> getMirror(p.getValue()))
    .then(p -> p.getValue().openStream());
```

The chain of Promises will also propagate any exceptions that occur to resolve the last Promise in the chain which means we do not need to do any exception handling in the intermediate tasks.

Monad

The Promise API supports monadic programming. See [4] Monad. The `Promise` interface defines a number of interesting methods including `map`, `flatMap` and `filter`.

- `filter(Predicate)` - Filter the value of the Promise.

  If the Promise is successfully resolved, the predicate argument is called with the value of the Promise. If the predicate accepts the value, then the value is used to successfully resolve the Promise returned by the filter method. If the predicate does not accept the value, the Promise returned by the filter method is unsuccessfully resolved with a No Such Element Exception. If the predicate throws an exception, the Promise returned by the filter method is unsuccessfully resolved with that exception.
If the Promise is unsuccessfully resolved, the predicate argument is not called and the Promise returned by the filter method is unsuccessfully resolved with the failure of the Promise.

- **map(Function)** - Map the value of the Promise.
  
  If the Promise is successfully resolved, the function argument is called with the value of the Promise. The value returned by the function is used to successfully resolve the Promise returned by the map method. If the function throws an exception, the Promise returned by the map method is unsuccessfully resolved with that exception.

  If the Promise is unsuccessfully resolved, the function argument is not called and the Promise returned by the map method is unsuccessfully resolved with the failure of the Promise.

- **flatMap(Function)** - FlatMap the value of the Promise.
  
  If the Promise is successfully resolved, the function argument is called with the value of the Promise. The Promise returned by the function is used to resolve the Promise returned by the flatMap method. If the function throws an exception, the Promise returned by the flatMap method is unsuccessfully resolved with that exception.

  If the Promise is unsuccessfully resolved, the function argument is not called and the Promise returned by the flatMap method is unsuccessfully resolved with the failure of the Promise.

- **recover(Function)** - Recover from the unsuccessful resolution of the Promise with a recovery value.
  
  If the Promise is successfully resolved, the function argument is not called and the Promise returned by the recover method is resolved with the value of the Promise.

  If the Promise is unsuccessfully resolved, the function argument is called with the Promise to supply a recovery value. If the recovery value is not null, the Promise returned by the recover method is successfully resolved with the recovery value. If the recovery value is null, the Promise returned by the recover method is unsuccessfully resolved with the failure of the Promise. If the function throws an exception, the Promise returned by the recover method is unsuccessfully resolved with that exception.

- **recoverWith(Function)** - Recover from the unsuccessful resolution of the Promise with a recovery Promise.
  
  If the Promise is successfully resolved, the function argument is not called and the Promise returned by the recover method is resolved with the value of the Promise.

  If the Promise is unsuccessfully resolved, the function argument is called with the Promise to supply a recovery Promise. If the recovery Promise is not null, the Promise returned by the recover method is resolved with the recovery Promise. If the recovery Promise is null, the Promise returned by the recover method is unsuccessfully resolved with the failure of the Promise. If the function throws an exception, the Promise returned by the recover method is unsuccessfully resolved with that exception.

- **fallbackTo(Promise)** - Fall back to the value of the Promise argument if the Promise unsuccessfully resolves.
  
  If the Promise is successfully resolved, the Promise argument is not used and the Promise returned by the fallbackTo method is resolved with the value of the Promise.

  If the Promise is unsuccessfully resolved, the Promise argument is used to provide a fallback value when it becomes resolved. If the Promise argument is successfully resolved, the Promise returned by the fallbackTo method is resolved with the value of the Promise argument. If the Promise argument is unsuccessfully resolved, the Promise returned by the fallbackTo method is unsuccessfully resolved with the failure of the Promise.

These functions can be used to build pipelines of chained Promises that are processed in sequence. For example, in the following chain, the value of the original promise, once resolved, is filtered for
acceptable values. If the filter says the value is not acceptable, the recover method will be used to re-
place it with a default value.

```java
return promise.filter(v -> isValueOk(v)).recover(p -> getDefaultValue())
```

With these chains, one can write powerful programs without the need to resort to complex if/else
and try/catch logic.

### 705.7 Functional Interfaces

In Java 8, the concept of Functional Interfaces is introduced. See [5] *Function Interfaces*. Functional
interfaces are interfaces with a single abstract method. Instances of functional interfaces can be
created with lambda expressions, method references, or constructor references. Many methods on
Promise take functional interface arguments and so are suitable for use with lambda expressions
and method references in Java 8.

Two of these functional interfaces are `Function` and `Predicate`. These are equivalent to functional in-
terfaces which are part of the `java.util.function` package introduced in Java 8. Since OSGi intends the
Promise API to be usable on versions of Java prior to Java 8, we define our own interfaces. In the fu-
ture, if Java 8 or later, becomes the base supported Java level for this specification, OSGi can update
the Promise interface to add default methods which accept the `java.util.function` versions of these
functional interfaces.

### 705.8 Promises Class

The Promises class provides several useful static methods when working with Promises.

Often, you may need to create an already resolved Promise to return or chain with another Promise.
The `resolved(T)` method can be used to create a new Promise already successfully resolved with the
specified value. The `failed(Throwable)` method can be used to create a new Promise already unsuc-
cessfully resolved with the specified exception.

```java
return getTimeConsumingAnswer().fallbackTo(Promises.resolved("Fallback Value"));
```

The Promises class also provides the `all(Promise,...)` and `all(Collection)` methods which return a
new Promise that is a latch on the specified Promises. The returned Promise must resolve only when
all of the specified Promises have resolved.

### 705.9 Security

The Promise API does not define any OSGi services nor does the API perform any privileged actions.
Therefore, it has no security considerations.

### 705.10 org.osgi.util.promise

Promise Package Version 1.0.

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

Example import for consumers using the API in this package:
705.10.1 Summary

- Deferred - A Deferred Promise resolution.
- FailedPromisesException - Promise failure exception for a collection of failed Promises.
- Failure - Failure callback for a Promise.
- Promise - A Promise of a value.
- Promises - Static helper methods for Promises.
- Success - Success callback for a Promise.

705.10.2 public class Deferred<T>

<T> The value type associated with the created Promise.

A Deferred Promise resolution.

Instances of this class can be used to create a Promise that can be resolved in the future. The associated Promise can be successfully resolved with resolve(Object) or resolved with a failure with fail(Throwable). It can also be resolved with the resolution of another promise using resolveWith(Promise).

The associated Promise can be provided to any one, but the Deferred object should be made available only to the party that will responsible for resolving the Promise.

Concurrency Immutable

705.10.2.1 public Deferred()

Create a new Deferred with an associated Promise.

705.10.2.2 public void fail(Throwable failure)

failure The failure of the resolved Promise. Must not be null.

Fail the Promise associated with this Deferred.

After the associated Promise is resolved with the specified failure, all registered callbacks are called and any chained Promises are resolved.

Resolving the associated Promise happens-before any registered callback is called. That is, in a registered callback, Promise.isDone() must return true and Promise.getValue() and Promise.getFailure() must not block.

Throws IllegalStateException – If the associated Promise was already resolved.

705.10.2.3 public Promise<T> getPromise()

Returns the Promise associated with this Deferred.

Returns The Promise associated with this Deferred.

705.10.2.4 public void resolve(T value)

value The value of the resolved Promise.

Successfully resolve the Promise associated with this Deferred.

After the associated Promise is resolved with the specified value, all registered callbacks are called and any chained Promises are resolved.
Resolving the associated Promise happens-before any registered callback is called. That is, in a registered callback, Promise.isDone() must return true and Promise.getValue() and Promise.getFailure() must not block.

Throws

IllegalStateException – If the associated Promise was already resolved.

705.10.2.5

public Promise<Void> resolveWith(Promise<? extends T> with)

with

A Promise whose value or failure must be used to resolve the associated Promise. Must not be null.

Resolve the Promise associated with this Deferred with the specified Promise.

If the specified Promise is successfully resolved, the associated Promise is resolved with the value of the specified Promise. If the specified Promise is resolved with a failure, the associated Promise is resolved with the failure of the specified Promise.

After the associated Promise is resolved with the specified Promise, all registered callbacks are called and any chained Promises are resolved.

Resolving the associated Promise happens-before any registered callback is called. That is, in a registered callback, Promise.isDone() must return true and Promise.getValue() and Promise.getFailure() must not block.

Returns

A Promise that is resolved only when the associated Promise is resolved by the specified Promise. The returned Promise must be successfully resolved with the value null, if the associated Promise was resolved by the specified Promise. The returned Promise must be resolved with a failure of IllegalStateException, if the associated Promise was already resolved when the specified Promise was resolved.

705.10.3

public class FailedPromisesException extends RuntimeException

Promise failure exception for a collection of failed Promises.

705.10.3.1

public FailedPromisesException(Collection<Promise<?>> failed, Throwable cause)

failed

A collection of Promises that have been resolved with a failure. Must not be null, must not be empty and all of the elements in the collection must not be null.

cause

The cause of this exception. This is typically the failure of the first Promise in the specified collection.

Create a new FailedPromisesException with the specified Promises.

705.10.3.2

public Collection<Promise<?>> getFailedPromises()

Returns the collection of Promises that have been resolved with a failure. The returned collection is unmodifiable.

705.10.4

public interface Failure

Failure callback for a Promise.

A Failure callback is registered with a Promise using the Promise.then(Success, Failure) method and is called if the Promise is resolved with a failure.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

Concurrency

Thread-safe

705.10.4.1

public void fail(Promise<?> resolved) throws Exception

resolved

The failed resolved Promise.
Failure callback for a Promise.

This method is called if the Promise with which it is registered resolves with a failure.

In the remainder of this description we will refer to the Promise returned by Promise.then(Success, Failure) when this Failure callback was registered as the chained Promise.

If this methods completes normally, the chained Promise must be failed with the same exception which failed the resolved Promise. If this method throws an exception, the chained Promise must be failed with the thrown exception.

**Throws** Exception – The chained Promise must be failed with the thrown exception.

### 705.10.5 public interface Promise<T>

<T> The value type associated with this Promise.

A Promise of a value.

A Promise represents a future value. It handles the interactions for asynchronous processing. A Deferred object can be used to create a Promise and later resolve the Promise. A Promise is used by the caller of an asynchronous function to get the result or handle the error. The caller can either get a callback when the Promise is resolved with a value or an error, or the Promise can be used in chaining. In chaining, callbacks are provided that receive the resolved Promise, and a new Promise is generated that resolves based upon the result of a callback.

Both callbacks and chaining can be repeated any number of times, even after the Promise has been resolved.

Example callback usage:

```java
final Promise<String> foo = foo();
foo.onResolve(new Runnable() {
    public void run() {
        System.out.println(foo.getValue());
    }
});
```

Example chaining usage:

```java
Success<String,String> doubler = new Success<String,String>() {
    public Promise<String> call(Promise<String> p) throws Exception {
        return Promises.resolved(p.getValue()+p.getValue());
    }
};
final Promise<String> foo = foo().then(doubler).then(doubler);
foo.onResolve(new Runnable() {
    public void run() {
        System.out.println(foo.getValue());
    }
});
```

**Concurrency** Thread-safe

**Provider Type** Consumers of this API must not implement this type

### 705.10.5.1 public Promise<T> fallbackTo(Promise<? extends T> fallback)

*fallback* The Promise whose value must be used to resolve the returned Promise if this Promise resolves with a failure. Must not be null.

- Fall back to the value of the specified Promise if this Promise fails.
If this Promise is successfully resolved, the returned Promise must be resolved with the value of this Promise.

If this Promise is resolved with a failure, the successful result of the specified Promise is used to resolve the returned Promise. If the specified Promise is resolved with a failure, the returned Promise must be failed with the failure of this Promise rather than the failure of the specified Promise.

This method may be called at any time including before and after this Promise has been resolved.

Returns A Promise that returns the value of this Promise or falls back to the value of the specified Promise.

705.10.5.2 public Promise<T> filter(Predicate<?> predicate)

predicate

The Predicate to evaluate the value of this Promise. Must not be null.

□ Filter the value of this Promise.

If this Promise is successfully resolved, the returned Promise must either be resolved with the value of this Promise, if the specified Predicate accepts that value, or failed with a NoSuchElementException, if the specified Predicate does not accept that value. If the specified Predicate throws an exception, the returned Promise must be failed with the exception.

If this Promise is resolved with a failure, the returned Promise must be failed with that failure.

This method may be called at any time including before and after this Promise has been resolved.

Returns A Promise that filters the value of this Promise.

705.10.5.3 public Promise<R> flatMap(Function<?,Promise<? extends R>> mapper)

Type Arguments <R>

The value type associated with the returned Promise.

mapper

The Function that must flatMap the value of this Promise to a Promise that must be used to resolve the returned Promise. Must not be null.

□ FlatMap the value of this Promise.

If this Promise is successfully resolved, the returned Promise must be resolved with the Promise from the specified Function as applied to the value of this Promise. If the specified Function throws an exception, the returned Promise must be failed with the exception.

If this Promise is resolved with a failure, the returned Promise must be failed with that failure.

This method may be called at any time including before and after this Promise has been resolved.

Returns A Promise that returns the value of this Promise as mapped by the specified Function.

705.10.5.4 public Throwable getFailure() throws InterruptedException

□ Returns the failure of this Promise.

If this Promise is not resolved, this method must block and wait for this Promise to be resolved before completing.

If this Promise was resolved with a failure, this method returns with the failure of this Promise. If this Promise was successfully resolved, this method must return null.

Returns The failure of this resolved Promise or null if this Promise was successfully resolved.

Throws InterruptedException – If the current thread was interrupted while waiting.

705.10.5.5 public T getValue() throws InvocationTargetException, InterruptedException

□ Returns the value of this Promise.

If this Promise is not resolved, this method must block and wait for this Promise to be resolved before completing.
If this Promise was successfully resolved, this method returns with the value of this Promise. If this Promise was resolved with a failure, this method must throw an InvocationTargetException with the failure exception as the cause.

**Returns** The value of this resolved Promise.

**Throws** InvocationTargetException – If this Promise was resolved with a failure. The cause of the InvocationTargetException is the failure exception.

InterruptedException – If the current thread was interrupted while waiting.

### 705.10.5.6

**public boolean isDone()**

- □ Returns whether this Promise has been resolved.
  - This Promise may be successfully resolved or resolved with a failure.

**Returns** true if this Promise was resolved either successfully or with a failure; false if this Promise is unresolved.

### 705.10.5.7

**public Promise<? extends R> map(Function<? extends ? extends R, ? extends T> mapper)**

**Type Arguments**

- <R> The value type associated with the returned Promise.

- mapper The Function that must map the value of this Promise to the value that must be used to resolve the returned Promise. Must not be null.

- □ Map the value of this Promise.
  - If this Promise is successfully resolved, the returned Promise must be resolved with the value of specified Function as applied to the value of this Promise. If the specified Function throws an exception, the returned Promise must be failed with the exception.
  - If this Promise is resolved with a failure, the returned Promise must be failed with that failure.
  - This method may be called at any time including before and after this Promise has been resolved.

**Returns** A Promise that returns the value of this Promise as mapped by the specified Function.

### 705.10.5.8

**public Promise<T> onResolve(Runnable callback)**

- callback A callback to be called when this Promise is resolved. Must not be null.

- □ Register a callback to be called when this Promise is resolved.
  - The specified callback is called when this Promise is resolved either successfully or with a failure.
  - This method may be called at any time including before and after this Promise has been resolved.
  - Resolving this Promise happens-before any registered callback is called. That is, in a registered callback, isDone() must return true and getValue() and getFailure() must not block.
  - A callback may be called on a different thread than the thread which registered the callback. So the callback must be thread safe but can rely upon that the registration of the callback happens-before the registered callback is called.

**Returns** This Promise.

### 705.10.5.9

**public Promise<T> recover(Function<? extends Promise<?>, ? extends T> recovery)**

- recovery If this Promise resolves with a failure, the specified Function is called to produce a recovery value to be used to resolve the returned Promise. Must not be null.

- □ Recover from a failure of this Promise with a recovery value.
  - If this Promise is successfully resolved, the returned Promise must be resolved with the value of this Promise.
If this Promise is resolved with a failure, the specified Function is applied to this Promise to produce a recovery value.

- If the recovery value is not null, the returned Promise must be resolved with the recovery value.
- If the recovery value is null, the returned Promise must be failed with the failure of this Promise.
- If the specified Function throws an exception, the returned Promise must be failed with that exception.

To recover from a failure of this Promise with a recovery value of null, the recoverWith(Function) method must be used. The specified Function for recoverWith(Function) can return Promises.resolved(null) to supply the desired null value.

This method may be called at any time including before and after this Promise has been resolved.

Returns A Promise that resolves with the value of this Promise or recovers from the failure of this Promise.

```
705.10.5.10 public Promise<T> recoverWith(Function<Promise<?>,Promise<? extends T>> recovery)
```

recovery If this Promise resolves with a failure, the specified Function is called to produce a recovery Promise to be used to resolve the returned Promise. Must not be null.

- Recover from a failure of this Promise with a recovery Promise.

  If this Promise is successfully resolved, the returned Promise must be resolved with the value of this Promise.

  If this Promise is resolved with a failure, the specified Function is applied to this Promise to produce a recovery Promise.

  - If the recovery Promise is not null, the returned Promise must be resolved with the recovery Promise.
  - If the recovery Promise is null, the returned Promise must be failed with the failure of this Promise.
  - If the specified Function throws an exception, the returned Promise must be failed with that exception.

This method may be called at any time including before and after this Promise has been resolved.

Returns A Promise that resolves with the value of this Promise or recovers from the failure of this Promise.

```
705.10.5.11 public Promise<R> then(Success<?,? extends R> success,Failure failure)
```

Type Arguments <R> The value type associated with the returned Promise.

success A Success callback to be called when this Promise is successfully resolved. May be null if no Success callback is required. In this case, the returned Promise must be resolved with the value null when this Promise is successfully resolved.

failure A Failure callback to be called when this Promise is resolved with a failure. May be null if no Failure callback is required.

- Chain a new Promise to this Promise with Success and Failure callbacks.

  The specified Success callback is called when this Promise is successfully resolved and the specified Failure callback is called when this Promise is resolved with a failure.

This method returns a new Promise which is chained to this Promise. The returned Promise must be resolved when this Promise is resolved after the specified Success or Failure callback is executed. The result of the executed callback must be used to resolve the returned Promise. Multiple calls to this method can be used to create a chain of promises which are resolved in sequence.
If this Promise is successfully resolved, the Success callback is executed and the result Promise, if any, or thrown exception is used to resolve the returned Promise from this method. If this Promise is resolved with a failure, the Failure callback is executed and the returned Promise from this method is failed.

This method may be called at any time including before and after this Promise has been resolved. Resolving this Promise happens-before any registered callback is called. That is, in a registered callback, isDone() must return true and getValue() and getFailure() must not block.

A callback may be called on a different thread than the thread which registered the callback. So the callback must be thread safe but can rely upon that the registration of the callback happens-before the registered callback is called.

Returns A new Promise which is chained to this Promise. The returned Promise must be resolved after the specified Success or Failure callback, if any, is executed.

705.10.5.12 public Promise<R> then(Success<?,? extends R> success)

Type Arguments <R>

success A Success callback to be called when this Promise is successfully resolved. May be null if no Success callback is required. In this case, the returned Promise must be resolved with the value null when this Promise is successfully resolved.

Return Chain a new Promise to this Promise with a Success callback. This method performs the same function as calling then(Success, Failure) with the specified Success callback and null for the Failure callback.

Returns A new Promise which is chained to this Promise. The returned Promise must be resolved when this Promise is resolved after the specified Success, if any, is executed.

See Also then(Success, Failure)

705.10.6 public class Promises

Static helper methods for Promises.

Concurrency Thread-safe

705.10.6.1 public static Promise<List<T>> all(Collection<Promise<S>> promises)

Type Arguments <T,S extends T>

promises The Promises which must be resolved before the returned Promise must be resolved. Must not be null and all of the elements in the collection must not be null.

Returns A Promise that is resolved only when all the specified Promises are resolved. The returned Promise must be successfully resolved with a List of the values in the order of the specified Promises if all the specified Promises are successfully resolved. The List in the returned Promise is the property of the caller and is modifiable. The returned Promise must be resolved with a failure of FailedPromisesException if any of the specified Promises are resolved with a failure. The failure FailedPromisesException must contain all of the specified Promises which resolved with a failure.
public static Promise<List<T>> all(Promise<? extends T> ... promises)

Type Arguments

<T>
The value type associated with the specified Promises.

promises
The Promises which must be resolved before the returned Promise must be resolved. Must not be null and all of the arguments must not be null.

□ Create a new Promise that is a latch on the resolution of the specified Promises.
The new Promise acts as a gate and must be resolved after all of the specified Promises are resolved.

Returns
A Promise that is resolved only when all the specified Promises are resolved. The returned Promise must be successfully resolved with a List of the values in the order of the specified Promises if all the specified Promises are successfully resolved. The List in the returned Promise is the property of the caller and is modifiable. The returned Promise must be resolved with a failure of FailedPromisesException if any of the specified Promises are resolved with a failure. The failure FailedPromisesException must contain all of the specified Promises which resolved with a failure.

public static Promise<T> failed(Throwable failure)

Type Arguments

<T>
The value type associated with the returned Promise.

failure
The failure of the resolved Promise. Must not be null.

□ Create a new Promise that has been resolved with the specified failure.

Returns
A new Promise that has been resolved with the specified failure.

public static Promise<T> resolved(T value)

Type Arguments

<T>
The value type associated with the returned Promise.

value
The value of the resolved Promise.

□ Create a new Promise that has been resolved with the specified value.

Returns
A new Promise that has been resolved with the specified value.

public interface Success<T,R>

<T>
The value type of the resolved Promise passed as input to this callback.

<R>
The value type of the returned Promise from this callback.

Success callback for a Promise.
A Success callback is registered with a Promise using the Promise.then(Success) method and is called if the Promise is resolved successfully.
This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

Concurrency
Thread-safe

public Promise<R> call(Promise<T> resolved) throws Exception

resolved
The successfully resolved Promise.

□ Success callback for a Promise.

This method is called if the Promise with which it is registered resolves successfully.
In the remainder of this description we will refer to the Promise returned by this method as the returned Promise and the Promise returned by Promise.then(Success) when this Success callback was registered as the chained Promise.

If the returned Promise is null then the chained Promise must resolve immediately with a successful value of null. If the returned Promise is not null then the chained Promise must be resolved when the returned Promise is resolved.

*Returns*  The Promise to use to resolve the chained Promise, or null if the chained Promise is to be resolved immediately with the value null.

*Throws*  Exception—The chained Promise must be failed with the thrown exception.

### 705.11  *org.osgi.util.function*

Function Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle’s manifest.

Example import for consumers using the API in this package:

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

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.util.function; version="[1.0,1.1)"
```

#### 705.11.1 Summary

- Function  - A function that accepts a single argument and produces a result.
- Predicate  - A predicate that accepts a single argument and produces a boolean result.

#### 705.11.2  *public interface Function<T,R>*

< T >  The type of the function input.

< R >  The type of the function output.

A function that accepts a single argument and produces a result.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

*Concurrency*  Thread-safe

##### 705.11.2.1  *public R apply(T t)*

< T >  The type of the function input.

< R >  The type of the function output.

Applies this function to the specified argument.

*Returns*  The output of this function.

#### 705.11.3  *public interface Predicate<T>*

< T >  The type of the predicate input.

A predicate that accepts a single argument and produces a boolean result.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

*Concurrency*  Thread-safe
public boolean test(T t)

  t  The input to this predicate.

  Evaluates this predicate on the specified argument.

  Returns  true if the specified argument is accepted by this predicate; false otherwise.

References

[1]  JavaScript Promises


[3]  ECMAScript 6 drafts

https://en.wikipedia.org/wiki/Monad_%28functional_programming%29

[5]  Function Interfaces
End of Review Specification