DISTRIBUTION AND FEEDBACK LICENSE

DISTRIBUTION PACKAGE TITLE: OSGi Service Platform Release 4 Version 4.2 - Early Draft 3

DATE OF DISTRIBUTION: 7 March 2009

© 2009 OSGi Alliance

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance expressly reserves all rights not granted pursuant to this limited copyright license including termination of the license at will at any time.

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

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

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

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

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution.

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

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

This document is Early Draft 3 of the OSGi Service Platform Release 4 Version 4.2 specifications. As an early draft, it contains non-final specification work and it is not organized in the format normally associated with final release OSGi specifications. This document contains copies of OSGi design documents which either propose to modify existing published OSGi specifications from the OSGi Service Platform Release 4 Version 4.1 specification documents or propose new specifications to potentially be incorporated in the final OSGi Service Platform Release 4 Version 4.2 specification documents.

Since this early draft is not a complete specification document, the reader is expected to be familiar with OSGi Technology and the currently published OSGi Service Platform Release 4 Version 4.1 specification documents. The reader should refer to http://www.osgi.org/About/Technology for more information on the OSGi Technology. There the reader can find a description of the OSGi Technology, as well as links to whitepapers and the OSGi Service Platform Release 4 Version 4.1 specification documents, which are all available for download.

In an effort to make this early draft available as quickly as possible, it contains OSGi design documents (“RFCs”). These documents have been declassified by the OSGi Alliance so that they may be made available in this early draft. This early draft contains a majority of the design documents the OSGi expert groups currently anticipate will be incorporated into the final specification documents.

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

BJ Hargrave
Chief Technical Officer
OSGi Alliance
Abstract

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

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

Copyright © OSGi Alliance 2006. All Rights Reserved.

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance expressly reserves all rights not granted pursuant to this limited copyright license including termination of the license at will at any time.

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

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

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

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution.

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable, worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future (“Future Specification”), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification.

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

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

0.1 Table of Contents

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

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

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

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

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

5 Technical Solution....................................................................................................................... 9
  5.1 Entities............................................................................................................................ 9
  5.2 Overview........................................................................................................................ 11
    5.2.1 Resource.................................................................................................................. 12
    5.2.2 Capabilities.............................................................................................................. 13
    5.2.3 Requirements.......................................................................................................... 13
    5.2.4 Extends.................................................................................................................... 14
  5.3 Repository Admin............................................................................................................... 14
    5.3.1 Discovery................................................................................................................ 14
    5.3.2 Resolving............................................................................................................... 14
    5.3.3 Admin..................................................................................................................... 15
  5.4 Resolving............................................................................................................................ 15
  5.5 XML Schema..................................................................................................................... 15
    5.5.1 Namespace............................................................................................................... 15
    5.5.2 The XML Structure................................................................................................. 15
    5.5.3 Repository.............................................................................................................. 15
    5.5.4 Referral.................................................................................................................... 16
    5.5.5 Resource.................................................................................................................. 16
    5.5.6 Category.................................................................................................................. 16
### 6 Open Issues

- **6.1 Uses Constraint**
- **6.2 Query protocol**
- **6.3 Licensing**

### 7 Java Documentation

#### 7.1 Interface Capability
- `getVersion` ...
- `getProperties` ...

#### 7.2 Interface CapabilityProvider
- `getCapabilities` ...

#### 7.3 Interface Repository
- `getURL` ...
- `getResources` ...
- `getName` ...
- `getLastModified` ...

#### 7.4 Interface RepositoryAdmin
- `discoverResources` ...
- `resolver` ...
- `addRepository` ...
- `removeRepository` ...
- `listRepositories` ...
- `getResource` ...

#### 7.5 Class RepositoryPermission
- `RepositoryPermission` ...

#### 7.6 Interface Requirement
- `getName` ...
- `getFilter` ...
- `isMultiple` ...
- `isOptional` ...
- `isExtend` ...
- `getComment` ...
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 1.1.

Source code is shown in this typeface.
0.3 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>DEC 22 2005</td>
<td>Peter Kriens, Initial draft</td>
</tr>
<tr>
<td></td>
<td>MAR 16 2006</td>
<td>Peter Kriens, Prepared for release.</td>
</tr>
<tr>
<td>updated</td>
<td>FEB 20 2009</td>
<td>Hal Hildebrand, resurrected RFC from zombie status, added schema for OBR</td>
</tr>
<tr>
<td>schema</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Updated api</td>
<td>FEB 25 2009</td>
<td>Hal Hildebrand, updated the API to reflect current state of Felix OBR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>implementation</td>
</tr>
</tbody>
</table>

1 Introduction

1.1 Acknowledgements

This document is based on the excellent work done by Richard S. Hall with the Oscar Bundle Repository.

1.2 Introduction

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

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

OSGi specifications are being adopted at an increasing rate. The number of bundles available world wide is likely in the thousands, if not low ten thousands. Although many of these bundles are proprietary and not suitable for distribution, there are a large number of distributable bundles available. The current situation is that vendors have proprietary bundle repositories. However, in the open source community, the Oscar Bundle Repository allows end users to discover bundles using a command line tool that runs on any OSGi Framework.

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

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

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

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

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

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

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

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

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

3 Problem Description

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

4 Requirements

4.1 Functional

• Provide browsing access to a bundle repository via a web server

• Provide access to a bundle repository so that bundles can be directly installed after discovery

• Handle dependency resolution so that bundles can be deployed without generating errors

• Allow repositories to be linked, creating a federated repository

• Provide programmatic (service) access to the repository

4.1 Discovery

• Search bundles by keywords

• Search by category

• Provide filtering capabilities on execution environment

• Licensing conditions must be available before downloading the artifacts
4.1 Dependency Resolution

- Must be able to find bundles that can solve any unresolved requirements
- Must be able to provide a list of cooperative bundles.
- Cooperative capabilities must be possible to select by a bundle or to be offered by a provider.
- Must handle all the requirements/capabilities and their directives as defined in the OSGi R4 specifications

4.1 Security

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

4.1 Non Functional

- The repository must be able to scale to ten thousand bundles
- Compliant with other OSGi services
- Easy to use
- It must be possible to implement a repository with a simple file. That is, a server must not be required

1 Technical Solution

1.1 Entities

- Repository Admin – A service that provides access to a federation of repositories.
- Repository – Provides access to a set of resources that are defines in a repository file
- Resource – A description of a bundle or other artifact that can be installed on a device. A resource provides capabilities and requires capabilities of other resources or the environment.
- Capability – A named set of properties
- Requirement – An assertion on a resource’s capabilities.
- Extend – A resource can act as an extension to another resource.
- Resolver – An object that can be used to find dependent and extension resources, as well as install them.
1.1.1.1 Domain Object Model

- Repository File – An XML file that can be referenced by a URL. The content contains meta data of resources and referrals to other repository files. It can be a static file or generated by a server.
1.1.1.2 Service Model

The key architecture of the OSGi Repository is a generic description of a resource and its dependencies. A resource is a bundle, but can potentially also be something else, for example, a certificate or configuration file. The purpose of the resource description is to discover applicable resources and deploy these resources without causing install errors due to missing dependencies.

For this purpose, each resource description has a list of requirements on other resources or the environment, a list of capabilities that are used to satisfy the requirements, and a list of extends, which are used to extend the capabilities of other resources. This is depicted (except for the extend) in the following picture.

---

1.2 Overview

The key architecture of the OSGi Repository is a generic description of a resource and its dependencies. A resource is a bundle, but can potentially also be something else, for example, a certificate or configuration file. The purpose of the resource description is to discover applicable resources and deploy these resources without causing install errors due to missing dependencies.

For this purpose, each resource description has a list of requirements on other resources or the environment, a list of capabilities that are used to satisfy the requirements, and a list of extends, which are used to extend the capabilities of other resources. This is depicted (except for the extend) in the following picture.
1.2.1 Resource

The resource is identified by the following methods:

- **getName** – A name for the resource that is globally unique for the function of the resource. There can exist multiple resources with the same name but a different version. Two resources with the same name and version are considered to be identical. For a bundle, this is normally mapped to the Bundle-SymbolicName manifest header.

- **getVersion** – A version for the resource. This must be a version usable by the OSGi Framework version class. For a bundle this is mapped to the Bundle-Version manifest header.

- **getId** – A local repository admin which is a handle to the resource object. This can not be used as a persistent id. Use name + version + repository URL for this.

The resource can contain any user defined properties. The properties can be obtained with the getProperties method. Properties are case sensitive. The following properties are predefined:

1. **id** – The id of the resource. This id is also available from the getId method and is automatically managed by the repository; it is not possible to override this property.

2. **version** – The version of the resource, managed by the repository; it is not possible to override this property.

3. **name** – The name of the resource. Managed by the repository; it is not possible to override this property.

4. **license** – A URI to the license file. This element is derived from the Bundle-License manifest header.

5. **description** – A textual description of the bundle. This must be unformatted text. This element is derived from the Bundle-Description manifest header.

6. **documentation** – A URI to the documentation. This element is derived from the Bundle-DocURL manifest header.

7. **copyright** – A copyright statement. This element is derived from the Bundle-Copyright manifest header.

8. **source** – A URI to a source distribution of the resource. This element is derived from the Bundle-Source manifest header.

9. **size** – The size of the resource in bytes.

Property names must follow the rules for bundle symbolic names.

The type of the property can be one of the following:

- **string** – Java String object

- **version** – org.osgi.framework.Version

- **uri** – Java URI object

- **long** –

- **double** –
• set – A comma separated list of values. White space around the commas must be discarded. The values cannot contain commas.

Properties that are of a specific type are compared and filtered according to their type. For versions, this must included checking for version ranges. That is, version=(1,2] must match any version that lies in the range (1,2).

1.2.1 Capabilities

A capability is anything that can be described with a set of properties. Examples of capabilities are:

• A package export
• A service export
• A fragment host
• A bundle
• A certificate
• A configuration record
• A group
• An Execution Environment
• A Display type
• Memory size
• Accessories

Capabilities are named. The reason they are named is so that they can only be provided to requirements with the same name. This is necessary because a property from two capabilities could have different meanings but still use the same name. To prevent these name clashes, the capabilities (and the requirements that they can resolve) are named. This specification defines names necessary to handle the capability/requirements of the OSGi Bundle Manifest.

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

1.2.1 Requirements

A requirement expressed as a filter on a resource. Just like a capability, a requirement is named. The filter must only be matched to capabilities with the same name. A requirement matches a capability when its filter matches any of the properties defined in that capability.

The syntax of the filter is the OSGi filter syntax. A filter was chosen because it allows the specification of arbitrary complex assertions. The disadvantage is that a filter is more or less opaque for the software, making it harder to provide assistance to the end user why certain resources are included and other not. A requirement can therefore optionally contain a reason. A reason is a short description that is applicable when a requirement is the cause for the selection of a resource.
For example, a package import is translated to a requirement for a package capability. If a bundle exports this package and it is selected, then the reason is the requirement for the import package.

Requirements can be optional and/or multiple.

<table>
<thead>
<tr>
<th>Optional</th>
<th>Multiple</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>1..1</td>
<td>One and only one solution is required.</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>1..*</td>
<td>At least one solution is required, but multiple solutions are useful</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>0..1</td>
<td>Optional, zero or one solution is required.</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>0..*</td>
<td>Optional, but multiple solutions are useful</td>
</tr>
</tbody>
</table>

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

1.2.2 Extends

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

1.3 Repository Admin

The Repository Admin service provides access to one or more repository files. That is, it represents a repository of federations. The Repository Admin service must ensure that a given resource is included only once and handle any circular references between repositories.

1.3.1 Discovery

The federated repository can be searched with an OSGi filter string. This filter can use any of the resource properties. The return is an array of Resource objects.

A specific Resource object can be found with the getResource method. This method takes the repository local id as parameter.

1.3.2 Resolving

Resolving can be an iterative process that takes as input a set of bundles and delivers a set of required bundles and a set of optional bundles. A special Resolver object is used to simplify the API significantly. The resolving process is further described in 1.1.

The resolver method creates a new resolver.

1.3.3 Admin

The maintenance of set of included repositories is handled by the following methods:
• **addRepository** – Add a new repository from a URL. This method will read the repository and any referred repositories.

• **removeRepository** – Remove a repository and all its referred repositories.

• **listRepositories** – Provide an array of Repository objects. These are only the top level repositories, referred repositories are not visible.

### 1.1 Resolving

The Resolver is an object that takes as input a set of bundles that should be added to a system. From this set, it can calculate the set of required bundles, choosing appropriate bundle when necessary. It also tracks a set of optional bundles. Optional bundles can be added to the input list.

The resolver is a complicated process requiring difficult choices that likely require user intervention and/or policies. The implementation of the Resolver object can provide this intelligence.

The api of the resolver is

- void add(Resource resource) – Add a resource to the resolver’s set
- Requirement[] getUnsatisfiedRequirements() - Answer the array of unsatisfied requirements on the current set of resources
- Resource[] getOptionalResources() - Answer the array of optional resources for the current state of resolution
- Requirement[] getReason(Resource resource) – Answer the array of requirements that led to the resolution of the resource
- Resource[] getResources(Requirement requirement) – Answer the array of resources resolved by the supplied requirement
- Resource[] getRequiredResources() - Answer the array of required resources for the resolution state
- Resource[] getAddedResources() - Answer the array of resources that were directly added to the resolver
- boolean resolve() - Attempt to resolve the currently defined set of resources, answer true if successful
- void deploy(boolean start) – Deploy the resolved set of resources. If “start” is true, then start the deployed bundles.

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

1.2.1 Namespace

The XML namespace is:

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

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

1.2.2 The XML Structure

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

repository ::= (referral | resource) *
resource ::= ANY * category * require * extend * capability *
capability ::= p *

1.2.3 Repository

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

1. name – The name of the repository. The name may contain spaces and punctuation.

2. time – The time the repository file was created. Time must be in YYYYMMDDHHmmSS.FFF. Where YYYY
is 4 digits for the year [2005,∞], MM is the 2 digit number of the month in the Gregorian Calendar
[1..12], DD the 2 digit number of the day in the month [1..31], HH is the 2 digit hour of the day in 24 hour
format [00..23], mm is the 2 digit number of minutes [00..59], SS are the 2 digit number of seconds
[0..59], and FFF is the 3 digit fraction of a second [000,999].

The repository element can only contain referral and resource elements.

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

1.2.1 Referral

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

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

2. url – The URL to the referred repository. The URL can be absolute or relative from the given repository’s
URL.

For example:

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

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

1. **name** – The name of the resource. In case of a bundle, this is the Bundle Symbolic Name.

2. **version** – The version of the resource. Version must follow the major, minor, micro, qualifier format as used the Framework’s version class.

The elements of the resource element can use arbitrarily named elements. These elements can use any tag name but must put the value in the text part of an element. Elements must not be repeated. The element may contain the following attribute:

1. **type** – One of the type strings given in: ###. The default is String. URI's are relative to the repository file.

For example:

```xml
<source type="uri">http://www.aqute.biz/bundles/console.src.jar</source>
```

1.2.1 Category

The <category> element defines a category. The purpose is to easy the discovery. Multiple category elements may be be provided. The category element has the following attributes:

- **id** – The id of the category.

For example:

```xml
<category id="osgi"/>
<category id="test"/>
```

1.2.1 Require

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

- **name** – The name of the requirement. The filter must only be applied to capabilities that have the same name.

- **filter** – The filter expression. The syntax must follow the OSGi filter syntax. The filter must correctly compare versions.

- **multiple** – If this requirements selects more than one candidate, then this is useful. The value is true or false.

- **optional** – If this requirement is necessary to satisfy the resource. The value is true or false.

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

For example:

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

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

### 1.2.1 Extend

The `<extend>` element is used for cooperative resources. A resource can "offer" itself to another resource as a useful cooperation. For example, a fragment with native code for a specific environment can offer itself to a host bundle. The extend element has exactly the same syntax as the requirement element. If this requirement matches a capability, then the resource of that capability is extended with the given resource.

For example:

```xml
<extend optional='false' multiple='false' name='bundle' filter='(&(symbolicname=org.eclipse.core.resources)(version&gt;=0.0.0))'>
    Required host for Fragment
</extend>
```

This example is for a fragment that belongs to the bundle with the symbolic name `org.eclipse.core.resources`. If the `org.eclipse.core.resources` bundle is selected to be deployed, then the given fragment must be offered for inclusion.

### 1.2.2 Capability

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

- **name** – Name of the capability. Only requirements with the same name must be able to match this capability.

Only the `<p>` element is allowed to be contained in the capability element. The `<p>` element has the following attributes:

- **n** – The name of the property
- **v** – The value of the property
- **t** – The type of the property. This must be one of:
  - string – A string value, which is the default.
  - version – An OSGi version as implemented in the OSGi Version class.
  - uri – A URI
  - long –
  - double –
  - set – A comma separated list of values. White space must be discarded, the values can not contain commas.

The following example shows a package export:
1.1 Filter Extensions

The OSGi filter language is based on LDAP. For this specification, the filter is extended with new capabilities.

1.1.1 Greater and Less Operators

The filter supports now all comparison operators: <, >, >=, <=. The absence of the < and > operators should have been fixed in R4.

1.1.2 Set Arithmetic

The filter must support SUBSET and SUPERSET capabilities. The set operators are:

- `key *> 1,2`        SUPERSET            <key> must contain at least 1 and 2 but may contain more.
- `key <* 1,2`        SUBSET            All of <key> must be in {1,2}. For a single property, this is a member test.

The value part of the filter must use a comma separated list of tokens. White-space must be ignored around the commas. The value must not contain a comma. If the property is a collection, the appropriate action is clear. If the property is a single value, it is translated to a set with a single element before the operator is executed.

If the value does not exist, then it is still possible to match a subset. A non-existent property is a proper subset of any set. A non-existent property is a superset if the list is empty.

- `(mandatory:<*vendor,var)`         Mandatory must contain vendor, var, both or be empty.

1.1.3 Version Ranges

The filter must support range checking for filters. The range syntax is equal to the Version range defined in the OSGi Manifest for Import-Package and Require-Bundle. If open ranges are used, the parentheses must be escaped with a backslash (use 2 backslashes in a Java string). This match must only be used if the property is a version.

- `(version=(1,2])` does not match 1.0.0, matches 1.1, 2, 2.0.0.qualifier

1.2 Sample XML File

```xml
<repository name='Untitled' time='20051210072623.031'>
  <resource version='3.0.0' name='org.osgi.test.cases.tracker' uri='org.osgi.test.cases.tracker-3.0.0.jar'>
    <size>
      44405
    </size>
    <documentation>
      http://www.osgi.org/
    </documentation>
  </resource>
</repository>
```
1.3 Querying a Web Service Based Repository

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

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

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

- **category** – A category

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

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

For example

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

1.1 Bundle Manifest Header Mapping

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

Every bundle must include a ‘bundle’ capability with the following properties:

- **symbolicname** – Bundle Symbolic Name. Must be set, type string.
- **version** – Version, must be set, type version.
- **manifestversion** – Version of the Manifest. Must be set, type version.
- **fragment-attachment** – If the fragment-attachment directive on the Bundle-SymbolicName is set. One of “always”, “never”, “resolve-time”.
- **singleton** – If the singleton directive is set. True or false, string type.

Example:

```xml
<capability name='bundle'>
  <p v='1' n='manifestversion'/>
  <p v='aQute.eclipse.osgi' n='symbolicname'/>
  <p v='1.0.1' t='version' n='version'/>
</capability>
```

1.1.1 Import and Export Package: ‘package’

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

- The type name is ‘package’.
- The ‘package’ property must be the name of the package. This property must be set and of type string.
- The ‘version’ property is the version. This property must be set and of type string.
- Add bundle-symbolic-name and bundle-version attributes
- Remaining attributes should be added to the capability. The directives must be suffixed with a ‘:’.
- Mandatory attributes must be put in a ‘set’ typed property with the name mandatory:. If no mandatory attributes are defined, an empty property must be defined.

For example:

```xml
<capability name="package">
  <p v="org.osgi.test.cases.tracker" n="package" />
  <p v="0.0.0" t="version" n="version" />
  <p v='vendor.var' n='mandatory:' type='set'/>
</capability>
```

An Import-Package clause is mapped to a Requirement.

- The type name is ‘package’
The filter must assert:

- package – Name of the package, e.g. (package=org.osgi.framework)
- version – Version or version range (the filter supports the version range syntax). E.g. (version=[1,2])
- Any custom attributes for equality
- That the mandatory: attribute is a proper subset of the asserted custom attributes. E.g (mandatory:<* attr1, attr2, attr3).

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

```xml
<require optional='false' multiple='false' name="package"
    filter="(&(package=org.osgi.test.cases.util)(version=1.1.0))"/>
```

### 1.1.1 Require-Bundle

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

- Type is ‘bundle’

Assert:

- symbolicname – The name of the bundle, e.g., (symbolicname=org.acme.xyx)
- version – Version range of the required bundle. (version=[1,2])

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

For example:

```xml
<require optional='false' multiple='false' name="bundle"
    filter="(&(symbolicname=org.eclipse.ui)(version>=0.0.0))"/>
```

### 1.1.1 Fragment-Host

The Fragment-Host is an Extend with the following filter assertions:

- symbolicname – The name of the bundle, e.g., (symbolicname=org.acme.xyx)
- version – Version range of the required bundle. (version=[1,2])

```xml
<extend optional='false' multiple='false' name="bundle"
    filter="(&(symbolicname=org.eclipse.core.resources)(version>=0.0.0))"/>
```

### 1.1.1 Import- and Export-Service

The Import-Service and Export-Service are deprecated, however, they are still useful for management purposes. Therefore, they are mapped to the generic requirement model.
Export-Service is mapped to a capability with the name ‘service’. The following properties are used.

- **service** – name of the service interface

For example

```xml
<capability name="service">
    <p v="com.ibm.wsn.resource.adapter.base.ResourceAdapterSubscriptionManagerIfc" n="service" />
    <p v="0.0.0" t="version" n="version" />
</capability>
```

Import-Service is mapped to an MULTIPLE requirement with the following assertions:

- **service** – Name of the service

```xml
<require optional='false' multiple='true' name="service"
    filter="(&(service=com.ibm.osg.webcontainer.WebContainer)(version=0.0.0))" />
```

1.1.1 Declarative Services

Declarative services also use the ‘service’ name. Each provided service interface must be listed as a capability. Each reference must be mapped to a requirement with the given cardinality (optional/multiple).

TBD.

1.1.2 Bundle-ExecutionEnvironment

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

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

```xml
<require optional='false' multiple='false' name="ee" filter="((ee=J2SE-1.4))" />
```

This requirement is UNARY.

2 Open Issues

2.1 Uses Constraint

The current specification does not address the uses directive on exported packages. The lack of handling this constraint makes it theoretically possible that a set of bundles is found that is resolved by the Repository resolver but can not be resolved by the Framework resolver. The following case demonstrates such a case:
A: import p;version=1, q;version=1
B: export p;version=1; uses:=q
    import q;version=2
C: export q;version=1
D: export q;version=2

A.p must be wired to B.p, however, B.q can only be wired to D.q which is not suitable for A. The uses constraint however requires B.q == A.q.

This issue should be further discussed. Maybe this is a generic problem that has a generic solution?

### 2.2 Query protocol

Richard thinks the query protocol is not necessary

### 2.3 Licensing

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

---

### 3 Java Documentation

#### 3.1 org.osgi.service.obr

**Interface Capability**

A named set of properties representing some capability that is provided by its owner.

**Method Summary**

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>getName()</td>
<td>Return the name of the capability.</td>
</tr>
<tr>
<td>java.util.Map</td>
<td>getProperties()</td>
<td>Return the set of properties.</td>
</tr>
</tbody>
</table>

**Method Detail**
3.1.1 getName

```java
public java.lang.String getName()
    Return the name of the capability.
```

3.1.2 getProperties

```java
public java.util.Map getProperties()
    Return the set of properties. Notice that the value of the properties is a list of values.
    Returns:
    a Map
```

3.2 org.osgi.service.obr

**Interface CapabilityProvider**

```java
public interface CapabilityProvider
```

This service interface allows third parties to provide capabilities that are present on the system but not encoded in the bundle's manifests. For example, a capability provider could provide:

1. A Set of certificates
2. Dimensions of the screen
3. Amount of memory
4. ...

### Method Summary

<table>
<thead>
<tr>
<th>Capability[]</th>
<th>getCapabilities()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Return a set of capabilities.</td>
</tr>
</tbody>
</table>

### Method Detail

3.2.1 getCapabilities

```java
public Capability[] getCapabilities()
```

Return a set of capabilities. These capabilities are considered part of the platform. Bundles can require these capabilities during selection. All capabilities from different providers are considered part of the platform.

Returns:
Set of capabilities
3.3 org.osgi.service.obr

Interface Repository

public interface Repository

Represents a repository.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>long <strong>lastModified</strong> ()</td>
<td>Return the last modified date of the repository.</td>
</tr>
<tr>
<td></td>
<td>java.lang.String <strong>getName</strong> ()</td>
<td>Return the name of this repository.</td>
</tr>
<tr>
<td></td>
<td>Resource[] <strong>getResources</strong> ()</td>
<td>Return the resources for this repository.</td>
</tr>
<tr>
<td></td>
<td>java.net.URL <strong>getURL</strong> ()</td>
<td>Return the associated URL for the repository.</td>
</tr>
</tbody>
</table>

### Method Detail

#### 3.3.1 getURL

```java
public java.net.URL getURL()
```

Return the associated URL for the repository.

#### 3.3.2 getResources

```java
public Resource[] getResources()
```

Return the resources for this repository.

#### 3.3.3 getName

```java
public java.lang.String getName()
```

Return the name of this repository. Returns a non-null name.

#### 3.3.4 getLastModified

```java
public long getLastModified()
```

Return the last modified date of the repository.
### Interface RepositoryAdmin

public interface RepositoryAdmin

Provides centralized access to the distributed repository. A repository contains a set of resources. A resource contains a number of fixed attributes (name, version, etc) and sets of:

1. Capabilities - Capabilities provide a named aspect: a bundle, a display, memory, etc.

2. Requirements - A named filter expression. The filter must be satisfied by one or more Capabilities with the given name. These capabilities can come from other resources or from the platform. If multiple resources provide the requested capability, one is selected. (### what algorithm? ###)

3. Requests - Requests are like requirements, except that a request can be fulfilled by 0..n resources. This feature can be used to link to resources that are compatible with the given resource and provide extra functionality. For example, a bundle could request all its known fragments. The UI associated with the repository could list these as optional downloads.

#### Method Summary

<table>
<thead>
<tr>
<th>Method (Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource[] addRepository(java.net.URL repository)</td>
<td>Add a new repository to the federation.</td>
</tr>
<tr>
<td>Resource[] discoverResources(java.lang.String filterExpr)</td>
<td>Discover any resources that match the given filter.</td>
</tr>
<tr>
<td>Resource getResource(java.lang.String repositoryId)</td>
<td></td>
</tr>
<tr>
<td>Repository[] listRepositories()</td>
<td>List all the repositories.</td>
</tr>
<tr>
<td>boolean removeRepository(java.net.URL repository)</td>
<td></td>
</tr>
<tr>
<td>Resolver resolver()</td>
<td>Create a resolver.</td>
</tr>
</tbody>
</table>

#### Method Detail

### 3.4.1 discoverResources

public Resource[] discoverResources(java.lang.String filterExpr) 
Discover any resources that match the given filter. This is not a detailed search, but a first scan of applicable resources. ### Checking the capabilities of the filters is not possible because that requires a
new construct in the filter. The filter expression can assert any of the main headers of the resource. The
attributes that can be checked are:

1. name
2. version (uses filter matching rules)
3. description
4. category
5. copyright
6. license
7. source

Parameters:
filterExpr - A standard OSGi filter

Returns:
List of resources matching the filters.

3.4.1 resolver

public Resolver resolver()
Create a resolver.
Returns:

3.4.2 addRepository

public Repository addRepository(java.net.URL repository)
throws java.lang.Exception
Add a new repository to the federation. The url must point to a repository XML file.
Parameters:
repository -
Returns:
Throws:
java.lang.Exception

3.4.3 removeRepository

public boolean removeRepository(java.net.URL repository)

3.4.4 listRepositories

public Repository[] listRepositories()
List all the repositories.
Returns:

3.4.5 getResource

public Resource getResource(java.lang.String repositoryId)
3.5 org.osgi.service.obr

Class RepositoryPermission

java.lang.Object
does not implement Serializable
java.security.Permission
does not implement Serializable
java.security.BasicPermission
does not implement Serializable
org.osgi.service.obr.RepositoryPermission

All Implemented Interfaces:
java.security.Guard, java.io.Serializable

public class RepositoryPermission
extends java.security.BasicPermission

TODO Implement

See Also:
Serialized Form

Constructor Summary

RepositoryPermission(java.lang.String name)

Methods inherited from class java.security/basicPermission

equals, getActions, hashCode, implies, newPermissionCollection

Methods inherited from class java.security.Permission

checkGuard, getName, toString

Methods inherited from class java.lang.Object

cloning, finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

3.5.1 RepositoryPermission

class RepositoryPermission

public RepositoryPermission(java.lang.String name)
## 3.6 org.osgi.service.obr

### Interface Requirement

public interface Requirement

A named requirement specifies the need for certain capabilities with the same name.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.String getName()</code></td>
<td></td>
<td>Return the name of the requirement.</td>
</tr>
<tr>
<td><code>java.lang.String getFilter()</code></td>
<td></td>
<td>Return the filter.</td>
</tr>
<tr>
<td><code>java.lang.String getComment()</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>boolean isMultiple()</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>boolean isOptional()</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>boolean isExtend()</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>boolean isSatisfied(Capability capability)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Method Detail

#### 3.6.1 getName

public `java.lang.String `getName()

Return the name of the requirement.

#### 3.6.2 getFilter

public `java.lang.String `getFilter()

Return the filter.

#### 3.6.3 isMultiple

public `boolean `isMultiple()

#### 3.6.4 isOptional

public `boolean `isOptional()
3.6.5 isExtend

public boolean isExtend()

3.6.6 getComment

public java.lang.String getComment()

3.6.7 isSatisfied

public boolean isSatisfied(Capability capability)

3.7 org.osgi.service.obr

Interface Resolver

public interface Resolver

Method Summary

void add(Resource resource)

void deploy(boolean start)

Resource[] getAddedResources()

Resource[] getOptionalResources()

Requirement[] getReason(Resource resource)

Resource[] getRequiredResources()

Resource[] getResources(Requirement requirement)

Requirement[] getUnsatisfiedRequirements()

boolean resolve()
Method Detail

3.7.1 add

```java
public void add(Resource resource)
    Add the resource to the list of resources to be resolved.
```

3.7.2 getUnsatisfiedRequirements

```java
public Requirement[] getUnsatisfiedRequirements()
    Answer the array of unsatisfied requirements of the current resolution state.
```

3.7.3 getOptionalResources

```java
public Resource[] getOptionalResources()
    Return the array of optional resources available for the resolution
```

3.7.4 getReason

```java
public Requirement[] getReason(Resource resource)
    Return the array of requirements that led to the resolution of the resource.
```

3.7.5 getResources

```java
public Resource[] getResources(Requirement requirement)
    Answer the array of resources resolved by the indicated requirement.
```

3.7.6 getRequiredResources

```java
public Resource[] getRequiredResources()
    Return the array of required resources for the resolution state.
```

3.7.7 getAddedResources

```java
public Resource[] getAddedResources()
    Return the array of resources manually added to the resolution.
```

3.7.8 resolve

```java
public boolean resolve()
    Resolve the required and optional resources derived from the supplied set of
c    resources. Answer true if the operation was successful.
```

3.7.9 deploy

```java
public void deploy(boolean start)
    Start the resolved resources of the resolver.
```
3.8 org.osgi.service.obr

Interface Resource

class interface Resource

A resource is an abstraction of a downloadable thing, like a bundle. Resources have capabilities and requirements. All a resource's requirements must be satisfied before it can be installed.

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPYRIGHT</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>DOCUMENTATION_URL</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>ID</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>KEYS</td>
<td>java.lang.String[]</td>
</tr>
<tr>
<td>LICENSE_URL</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>PRESENTATION_NAME</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>SIZE</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>SOURCE_URL</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>SYMBOLIC_NAME</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>URL</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>VERSION</td>
<td>java.lang.String</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCapabilities()</td>
<td>Capability[]</td>
</tr>
<tr>
<td>getCategories()</td>
<td>java.lang.String[]</td>
</tr>
<tr>
<td>getId()</td>
<td>java.lang.String</td>
</tr>
</tbody>
</table>
### Field Detail

#### 3.8.1 LICENSE_URL

```java
public static final java.lang.String LICENSE_URL
```
See Also:  
[Constant Field Values](#)

#### 3.8.2 DESCRIPTION

```java
public static final java.lang.String DESCRIPTION
```
See Also:  
[Constant Field Values](#)

#### 3.8.3 DOCUMENTATION_URL

```java
public static final java.lang.String DOCUMENTATION_URL
```
See Also:  
[Constant Field Values](#)

#### 3.8.4 COPYRIGHT

```java
public static final java.lang.String COPYRIGHT
```
See Also:  
[Constant Field Values](#)

#### 3.8.5 SOURCE_URL

```java
public static final java.lang.String SOURCE_URL
```
See Also:  
[Constant Field Values](#)
3.8.6 SYMBOLIC_NAME

public static final java.lang.String SYMBOLIC_NAME
See Also:
  Constant Field Values

3.8.7 PRESENTATION_NAME

public static final java.lang.String PRESENTATION_NAME
See Also:
  Constant Field Values

3.8.8 ID

public static final java.lang.String ID
See Also:
  Constant Field Values

3.8.9 VERSION

public static final java.lang.String VERSION
See Also:
  Constant Field Values

3.8.10 URL

public static final java.lang.String URL
See Also:
  Constant Field Values

3.8.11 SIZE

public static final java.lang.String SIZE
See Also:
  Constant Field Values

3.8.12 KEYS

public static final java.lang.String[] KEYS

Method Detail

3.8.13 getProperties

public java.util.Map getProperties()

3.8.14 getSymbolicName

public java.lang.String getSymbolicName()

3.8.15 getPresentationName

public java.lang.String getPresentationName()
3.8.16 getVersion

public org.osgi.framework.Version getVersion()

3.8.17 getId

public java.lang.String getId()

3.8.18 getURL

public java.net.URL getURL()

3.8.19 getRequirements

public Requirement[] getRequirements()

3.8.20 getCapabilities

public Capability[] getCapabilities()

3.8.21 getCategories

public java.lang.String[] getCategories()

3.8.22 getRepository

public Repository getRepository()

4 OBR Schema

<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.osgi.org/xmlns/obr/v1.0.0/"
  xmlns:tns="http://www.osgi.org/xmlns/obr/ v1.0.0/"
  elementFormDefault="qualified">
  <complexType name="Repository">
    <sequence>
      <choice maxOccurs="unbounded" minOccurs="0">
        <element ref="tns:resource"></element>
        <element ref="tns:referral"></element>
      </choice>
    </sequence>
    <attribute name="name" type="string">
    </attribute>
  </complexType>
</schema>
The name of the repository. The name may contain spaces and punctuation.

The time the repository file was created.

The elements of the resource element can use arbitrarily named elements. These elements can use any tag name but must put the value in the text part of an element. Elements must not be repeated.

The name of the resource. In case of a bundle, this is the Bundle Symbolic Name.
<complexType name="Referral">
  <annotation>
    <documentation>
      A referral points to another repository XML file. The purpose of this element is to create a federation of repositories that can be accessed as a single repository.
    </documentation>
  </annotation>
  <attribute name="depth" type="int">
    <annotation>
      <documentation>
        The depth of referrals this repository acknowledges.
      </documentation>
    </annotation>
  </attribute>
  <attribute name="url" type="anyURI">
    <annotation>
      <documentation>
        The URL to the referred repository. The URL can be absolute or relative from the given repository's URL.
      </documentation>
    </annotation>
  </attribute>
</complexType>

<complexType name="Category">
  <annotation>
    <documentation>
      The purpose is to ease the discovery. Multiple category elements may be may be provided.
    </documentation>
  </annotation>
  <attribute name="id" type="string">
    <annotation>
    </annotation>
  </attribute>
</complexType>

<complexType name="Resource">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Repository">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Referral">
  <attribute name="depth" type="int">
    <annotation>
      <documentation>
        The depth of referrals this repository acknowledges.
      </documentation>
    </annotation>
  </attribute>
  <attribute name="url" type="anyURI">
    <annotation>
      <documentation>
        The URL to the referred repository. The URL can be absolute or relative from the given repository's URL.
      </documentation>
    </annotation>
  </attribute>
<br>

<complexType name="Category">
  <attribute name="id" type="string">
    <annotation>
    </annotation>
  </attribute>
</complexType>

<complexType name="Resource">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Repository">
  <attribute name="uri" type="anyURI">\n  </attribute>
</complexType>

<complexType name="Referral">
  <attribute name="depth" type="int">
    <annotation>
      <documentation>
        The depth of referrals this repository acknowledges.
      </documentation>
    </annotation>
  </attribute>
  <attribute name="url" type="anyURI">
    <annotation>
      <documentation>
        The URL to the referred repository. The URL can be absolute or relative from the given repository's URL.
      </documentation>
    </annotation>
  </attribute>
<br>

<complexType name="Category">
  <attribute name="id" type="string">
    <annotation>
    </annotation>
  </attribute>
</complexType>

<complexType name="Resource">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Repository">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Referral">
  <attribute name="depth" type="int">
    <annotation>
      <documentation>
        The depth of referrals this repository acknowledges.
      </documentation>
    </annotation>
  </attribute>
  <attribute name="url" type="anyURI">
    <annotation>
      <documentation>
        The URL to the referred repository. The URL can be absolute or relative from the given repository's URL.
      </documentation>
    </annotation>
  </attribute>
<br>

<complexType name="Category">
  <attribute name="id" type="string">
    <annotation>
    </annotation>
  </attribute>
</complexType>

<complexType name="Resource">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Repository">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Referral">
  <attribute name="depth" type="int">
    <annotation>
      <documentation>
        The depth of referrals this repository acknowledges.
      </documentation>
    </annotation>
  </attribute>
  <attribute name="url" type="anyURI">
    <annotation>
      <documentation>
        The URL to the referred repository. The URL can be absolute or relative from the given repository's URL.
      </documentation>
    </annotation>
  </attribute>
<br>

<complexType name="Category">
  <attribute name="id" type="string">
    <annotation>
    </annotation>
  </attribute>
</complexType>

<complexType name="Resource">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Repository">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Referral">
  <attribute name="depth" type="int">
    <annotation>
      <documentation>
        The depth of referrals this repository acknowledges.
      </documentation>
    </annotation>
  </attribute>
  <attribute name="url" type="anyURI">
    <annotation>
      <documentation>
        The URL to the referred repository. The URL can be absolute or relative from the given repository's URL.
      </documentation>
    </annotation>
  </attribute>
<br>

<complexType name="Category">
  <attribute name="id" type="string">
    <annotation>
    </annotation>
  </attribute>
</complexType>

<complexType name="Resource">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Repository">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Referral">
  <attribute name="depth" type="int">
    <annotation>
      <documentation>
        The depth of referrals this repository acknowledges.
      </documentation>
    </annotation>
  </attribute>
  <attribute name="url" type="anyURI">
    <annotation>
      <documentation>
        The URL to the referred repository. The URL can be absolute or relative from the given repository's URL.
      </documentation>
    </annotation>
  </attribute>
<br>

<complexType name="Category">
  <attribute name="id" type="string">
    <annotation>
    </annotation>
  </attribute>
</complexType>

<complexType name="Resource">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Repository">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Referral">
  <attribute name="depth" type="int">
    <annotation>
      <documentation>
        The depth of referrals this repository acknowledges.
      </documentation>
    </annotation>
  </attribute>
  <attribute name="url" type="anyURI">
    <annotation>
      <documentation>
        The URL to the referred repository. The URL can be absolute or relative from the given repository's URL.
      </documentation>
    </annotation>
  </attribute>
<br>

<complexType name="Category">
  <attribute name="id" type="string">
    <annotation>
    </annotation>
  </attribute>
</complexType>

<complexType name="Resource">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>

<complexType name="Repository">
  <attribute name="uri" type="anyURI">
  </attribute>
</complexType>
<element name="category" type="tns:Category"></element>

<element name="require" type="tns:Require"></element>

<complexType name="Dependency">
  <attribute name="name" type="string">
    <annotation>
      <documentation>
        The filter must only be applied to capabilities that have the same name.
      </documentation>
    </annotation>
  </attribute>
  <attribute name="filter" type="tns:Filter">
    <annotation>
      <documentation>
        The filter expression. The syntax must follow the OSGi filter syntax. The filter must correctly compare versions.
      </documentation>
    </annotation>
  </attribute>
  <attribute name="multiple" type="boolean"></attribute>
  <attribute name="optional" type="boolean"></attribute>
</complexType>

<simpleType name="Filter">
  <annotation>
    <documentation>
      The syntax of a filter is described in the OSGi specification section 3.2.6 (p29 in R4).
    </documentation>
  </annotation>
  <restriction base="string"></restriction>
</simpleType>

<complexType name="Require">
  <annotation>
    <documentation>
      Describes one of the requirement that the enclosing resource has on its environment. A requirement is of a specific named type and contains a filter that is applied to all capabilities of the given type.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="tns:Dependency"></extension>
  </complexContent>
</complexType>

<complexType name="Extend">
  <annotation>
    <documentation>
      Used for cooperative resources. A resource can “offer” itself to another resource as a useful cooperation. For example, a fragment with native code for a specific environment can offer itself to a host bundle. The extend element has exactly the same syntax as the requirement element. If this requirement matches a capability, then the resource of that capability is extended with the given resource.
    </documentation>
  </annotation>
</complexType>
5 Security Considerations

5.1 Repository Permission
The Repository Permission protects the methods of the Repository Admin. The following actions are supported:

- **admin** – Protects addRepository and removeRepository.
- **browse** – Protects listRepositories, getResource, and searchResources
- **resolve** – Protects the Resolver method resolve
- **deploy** – Protects the Resolver method deploy

The name of this permission is irrelevant.

1 Document Support

1.1 References

1.1 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Peter Kriens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>aQute</td>
</tr>
<tr>
<td>Address</td>
<td>9c, Avenue St. Drezery, Beaulieu, France</td>
</tr>
<tr>
<td>Voice</td>
<td>+33 467542167</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:Peter.Kriens@aQute.biz">Peter.Kriens@aQute.biz</a></td>
</tr>
<tr>
<td>Name</td>
<td>Richard S. Hall</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Company</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Saginaw, Mi, USA</td>
</tr>
<tr>
<td>Voice</td>
<td>+1</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:Heavy@ungoverned.org">Heavy@ungoverned.org</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Hal Hildebrand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Oracle Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>500 Oracle Parkway, MS 2op946</td>
</tr>
<tr>
<td>Voice</td>
<td>+1 506 2055</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:hal.hildebrand@oracle.com">hal.hildebrand@oracle.com</a></td>
</tr>
</tbody>
</table>

1.2 Acronyms and Abbreviations

1.3 End of Document
Abstract

This document explores the complexity of running more than one OSGi framework in a single JVM process, and proposes a set of requirements to support that environment.
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................................................................................................................................6

2 Application Domain.....................................................................................................................6
  2.1 Behavior of multiple frameworks in one JVM.................................................................6
  2.2 Terminology + Abbreviations...........................................................................................7

3 Problem Description....................................................................................................................8
  3.1 Using multiple frameworks for isolation...........................................................................8
    3.1.1 Use case 1: Runtime/Application isolation.................................................................8
    3.1.2 Use case 2: Construction of composites......................................................................9
    3.1.3 Use case 3: Application scoping/isolation.................................................................9

4 Requirements...........................................................................................................................9

5 Technical Solution....................................................................................................................10
  5.1 Multi-framework environments.........................................................................................11
    5.1.1 Peer embedded frameworks.......................................................................................11
    5.1.2 Nested frameworks....................................................................................................11
    5.1.3 Parent/Child frameworks..........................................................................................11
    5.1.4 Identifying the properties used to start a framework..................................................12
  5.2 Managing a Child Framework - Provisional.................................................................13
    5.2.1 Composite and Surrogate Bundles................................................................................13
    5.2.2 Creating a child framework..........................................................................................14
  5.3 Sharing packages and services with composite bundles - Provisional.............................16
    5.3.1 Importing packages......................................................................................................16
    5.3.2 Exporting packages......................................................................................................17
    5.3.3 Composite resolution process.....................................................................................18
    5.3.4 Importing services........................................................................................................18
    5.3.5 Exporting services........................................................................................................18
  5.4 Singleton Multiplexing........................................................................................................19
    5.4.1 Multiplexing factories - Provisional..............................................................................19
    5.4.2 Other VM Singletons....................................................................................................20
    5.4.3 OSGi Singletons............................................................................................................20
  5.5 Open Issues.........................................................................................................................20
0.2 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 9.1.

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>AUG 12 2008</td>
<td>Initial Draft (boiler plate)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Erin Schnabel, IBM <a href="mailto:schnabel@us.ibm.com">schnabel@us.ibm.com</a></td>
</tr>
<tr>
<td>Additional text</td>
<td>SEP 05 2008</td>
<td>First pass at description of solution</td>
</tr>
<tr>
<td>Piscataway F2F</td>
<td>SEP 18 2008</td>
<td>Clarification of parent-child resource sharing (description of resources, relationships between the framework, etc.).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Glyn Normington, SpringSource, <a href="mailto:glyn.normington@springsource.com">glyn.normington@springsource.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Erin Schnabel, IBM <a href="mailto:schnabel@us.ibm.com">schnabel@us.ibm.com</a></td>
</tr>
<tr>
<td>Service changes</td>
<td>SEP 26 2008</td>
<td>Separation of concerns: FrameworkService for starting a child framework, a FrameworkBridgeService for sharing resources between frameworks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Erin Schnabel, IBM <a href="mailto:schnabel@us.ibm.com">schnabel@us.ibm.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Erin Schnabel, IBM <a href="mailto:schnabel@us.ibm.com">schnabel@us.ibm.com</a></td>
</tr>
<tr>
<td>API updates</td>
<td>NOV 04 2008</td>
<td>Clean up for public draft</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bug 829: using collections rather than arrays (bug 827), changing SystemBundle to Framework, method names on FrameworkFactory should start with &quot;new&quot;, clarify properties used for registration of link service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bug 828: description of service dynamics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Erin Schnabel, IBM <a href="mailto:schnabel@us.ibm.com">schnabel@us.ibm.com</a></td>
</tr>
<tr>
<td>API revision</td>
<td>NOV 11 2008</td>
<td>Reversion to strict parent-child relationship. Simplify to a single factory method: a child framework is always created with two link bundles, regardless of whether or not resources are shared.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Erin Schnabel, IBM <a href="mailto:schnabel@us.ibm.com">schnabel@us.ibm.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>
| API revision     | DEC 10 2008 | LinkBundle -> CompositeBundle  
bug 781: Changes to make FrameworkUtil.createFilter, AdminPermission, BundleSignerCondition framework agnostic.  
bug 863: AllPermission required to create a nested/child framework; framework implementation must ensure correct permissions on composite bundles.  
bug 881: New method on Bundle interface to get a bundle's signer DN chain.  
bug 905: BundleException if non-CompositeBundle already installed w/ provided location string; clarification of allowed manifest headers for composite bundles  
bug 906: first pass at concrete use cases for creating child frameworks |
| API finalization | DEC 30 2008 | Section 3.1 for isolation use cases (bug 906)  
API Finalization: CompositeBundle, SurrogateBundle, CompositeBundleFactory  
bug 813: Composite and surrogate bundle start level  
Erin Schnabel, IBM, schnabel@us.ibm.com |
| Additional revisions | JAN 12 2009 | bug 765: headers returned by the system bundle in getHeaders  
bug 814: sharing system bundle packages (section 5.3)  
bug 904: behavior of attributes and directives (section 5.3)  
bug 906: three concrete use cases for multiple frameworks in a VM (section 3.1)  
bug 909: singleton interface for multiplexing (section 5.4.1 + javadoc)  
Erin Schnabel, IBM, schnabel@us.ibm.com  
Tom Watson, IBM, tjwatson@us.ibm.com |
| post-CPEG        | JAN 13 2009 | Mark multiplexer API “Provisional” as well  
Constants for method names added to multiplexer API  
Remove “framework” from multiplexer package name  
Add static helper that provides basic implementation of reflection to check for osgi support and to maintain/traverse the double-linked list. |
1 Introduction

The OSGi platform is increasingly being used as a basis for composing complex runtime environments. Middleware providers are using OSGi implementations as the foundation of their runtimes, while application developers are looking to use OSGi standards and artifacts to build enterprise applications. As these two uses converge, questions arise over how to isolate applications from the runtime, or even certain applications from other applications.

This RFP has its origins in the CPEG conference call on 11 April, 2008. In that call a new RFC (RFC 136: “Scopes and OSGi”) was discussed for the first time, and RFC 132 (“Command Line Interface”) was reviewed. RFC 136 is a response to RFP 100 (“Visibility Scoping of Exported Packages”), and outlines a proposal for a defining visibility scopes within a framework. RFC 132 defines a common mechanism for booting both standalone and embedded OSGi frameworks. Nested frameworks (or even peer frameworks) in the same JVM were mentioned during the discussion of both RFCs: nested or peer frameworks provide an implicit scoping mechanism with additional isolation characteristics (RFC 136/RFP 100), and would be booted using those mechanisms defined in RFC 132.

2 Application Domain

As noted above, having multiple frameworks in a single JVM has been mentioned tangentially in relationship to several RFPs and RFCs. Some elements of the OSGi specification already apply to and would support multi-framework environments:

- A parent framework can provide classes to a nested framework using an existing property to specify what the nested framework’s system bundle must export\(\text{org.osgi.framework.system.packages}\). This property does not currently define the packages a parent framework provides. Instead it tells the framework what packages from the class loader that loaded the framework should be exported by the system bundle.

- RFC 132 defines a mechanism for setting properties for a particular framework instance. Without that mechanism, the BundleContext interface only provides a mechanism to get properties, not to set them.

- RFC 132 also provides a mechanism for controlling the state of embedded frameworks (start, stop), and defines rules for managing a framework’s lifecycle.

2.1 Behavior of multiple frameworks in one JVM

A framework instance is a discrete unit: it has its own class loading policy, and is responsible for resolving class loading constraints for all bundles that it “owns” (all bundles installed for that framework).

If there are multiple frameworks in a JVM:
There may be more than one instance of a singleton bundle active in the JVM: each framework manages its installed bundles independently from the other frameworks-- one framework should not have to know that a singleton bundle has been instantiated by another framework in the same JVM.

The Java mechanisms for extending the URL class allow handler factories to be specified once per JVM process. As bundles can have a much shorter life cycle than the JVM process as a whole, the URL Handlers Service Specification requires frameworks to register `URLStreamHandlerFactory` and `ContentHandlerFactory` objects to act as proxies for handlers that are registered with the framework as services. If multiple frameworks are present in a JVM, some mediation would have to occur to ensure that the single `URLStreamHandlerFactory` and `ContentHandlerFactory` can multiplex between available frameworks.

There is only one `Security Manager` reachable via the `java.lang.System` class. The Conditional Permission Admin (CPA) service interacts directly with the configured Security Manager to perform permission checks. The impact of any mechanism to adapt the Security Manager to a multi-framework environment on CPA condition evaluation would have to be well understood. At a base level, the discrete nature of each framework should be maintained: each framework should be able to establish, maintain, and evaluate its conditions, permissions and bundle protection domains independently and without knowledge of any other framework. Postponed conditions cannot be supported without multiplexing the Security Manager.

Issues with class loader parenting and use of the context class loader are not new, nor are they particular to a multi-framework environment.

### 2.2 Terminology + Abbreviations

- **Embedded framework**: An OSGi framework started from within an existing Java process.

- **Nested framework**: An embedded OSGi framework started by an OSGi bundle. The lifecycle of a nested framework is bounded by the lifecycle of the launching bundle (in other words, it is the launching bundle's responsibility to stop and clean up the framework when it is stopped).

- **Hosting framework**: An OSGi framework that contains other framework instances. Since nested frameworks (in general) are started by arbitrary bundles, the hosting framework may not be aware of nested instances.

- **Child framework**: A special case of nested framework-- an OSGi framework started by an OSGi framework. The lifecycle of a child framework is bounded by the lifecycle of the parent framework.

- **Parent framework**: A special case of hosting framework-- an OSGi framework that owns and manages child framework instances.

- **Peer framework**: An embedded framework at the same level of nesting as another embedded framework in a multi-framework environment: e.g. two nested frameworks started by the same bundle, or two child frameworks started by the same parent.

- **Multi-framework environment**: a JVM containing more than one framework, regardless of how those frameworks are related to each other.
3 Problem Description

Multiple frameworks cannot easily be present in the same JVM due to some of the constraints mentioned above:

- There isn't a uniform, framework-agnostic way to deal with JVM Singletons like the registered URLStream and ContentHandler factories for URLs. Some framework implementations (Eclipse Equinox, Apache Felix) have invented multiplexing handler factories that a) only handle frameworks of from the same vendor, b) use reflection to ensure the right handler factories are being used, and c) rely on their custom handler factories to mediate calls to factory methods between frameworks. There are synchronization and resource management issues with this approach (what happens when a framework is restarted, etc).

- There isn't a clear definition of how packages and/or services would or could be shared between multiple frameworks in the same JVM, or how managing the lifecycle of one framework would impact other frameworks that are using resources provided by that framework.

Standardizing a set of rules and expected behavior will allow multiple frameworks, potentially provided by different vendors, to behave and interact consistently when run within the same JVM.

3.1 Using multiple frameworks for isolation

As mentioned briefly in the introduction, using embedded frameworks provides an implicit scoping mechanism with strong isolation characteristics: without the creation of special mechanisms to share resources, each embedded framework provides an isolated scope for OSGi artifacts and resources (separate service registries, distinct class spaces, independent event notifications, etc.).

3.1.1 Use case 1: Runtime/Application isolation

There is a requirement for this kind of isolation in the enterprise environment: many enterprise middleware providers are using OSGi as a foundation for composable middleware runtimes. At the same time, enterprise application developers want to use OSGi to develop and manage the lifecycle of their applications. There are several issues caused by the middleware runtime and OSGi applications sharing the same OSGi framework:

- Accidental creation of API: Middleware providers often create infrastructure services meant only to support management and provisioning of the runtime itself. Because the API for these services are shared (and published to the OSGi registry), there isn't a good way to demarcate “internal” API from “external” API, short of extensive use of security (which has performance implications), or reliance on brittle package or bundle decorations, attribute matching, or framework-specific bundle/package resolution mechanisms.

- Conflicts over use of third party libraries: Some third party libraries may be required by both the middleware runtime and enterprise OSGi applications. Without some isolation between the runtime and applications, the runtime's use of third party libraries (and versioning requirements, etc.) could significantly impact and restrict the libraries available for use by OSGi applications.

This presents a use case for a nested framework approach: assume Framework A is an enterprise-level middleware runtime that is OSGi-based and composed from a set of proprietary bundles. The middleware provider could create a nested/child Framework B to host an arbitrary number of deployed applications, and should ideally be able to share with it a well-defined list of packages and services that are then available for use.
by the applications running in that inner framework. Aside from those explicitly shared resources, the two frameworks remain completely independent.

The child framework in this use case is long-lived, and the creator of the child framework (the parent middleware runtime) is not aware at creation time of the bundles that will be used by the child framework: as applications are installed or removed, the list of bundles installed in the child framework will change.

### 3.1.2 Use case 2: Construction of composites

Another use case for nested frameworks is the definition and use of “Composites”: the composite would declaratively specify the packages and services it would provide (as well as the packages and services it requires), but would otherwise provide isolation between the user of the composite, and the components providing the composite's implementation.

For example, a declarative composite definition could look something like this:

```plaintext
composite {
  Install-Bundle: foo; bundle-version=[1.0.0,1.2.0),
  bar; bundle-version=[1.0.0,1.0.0]
  Import-Package: waz; version=2.0.0
  Export-Package: baz; version=1.0.1
}
```

This composite could be implemented using a child framework: package waz would be shared with the child framework (just as with the first use case), but here, the child will also be exporting package baz back to the parent. From the point of view of other bundles in the parent, the composite should look like any other bundle that imports or exports packages or that uses or provides services. The parent bundles should have no awareness of what bundles the child is using to provide package baz. Bundles in the child framework, similarly, have no view of the bundles used by the parent framework to provide package waz.

Unlike the first use case, the composite is intended to provide packages and services to the hosting framework, and the full set of bundles that provide the composite implementation are known at declaration time.

### 3.1.3 Use case 3: Application scoping/isolation

Applications need to be isolated from each other. It should be possible to install multiple applications and be assured that they will not interact in unexpected ways (i.e. it is necessary to selectively prevent packages and services being resolved across applications). In the same way a runtime may want to have private infrastructure services and libraries, applications may also have services and libraries intended to be used/resolved internally; they may also depend on different/conflicting third-party library versions and therefore must execute in isolated frameworks in order for each application to resolve to the libraries it requires. The runtime/hosting environment may also provide packages and services to the isolated applications.

### 4 Requirements

1. The solution MUST define the parent class loader of child frameworks.
2. The solution MUST describe how the lifecycle of nested or embedded frameworks is managed and how peer, child or parent relationships are impacted by framework state changes.

3. The solution MUST allow frameworks to manage the lifecycle of their installed bundles independently from the lifecycle of bundles in other frameworks.

4. The solution MUST define a framework-neutral way to extend the URL Handlers Service Specification to address a multi-framework environment.

5. The solution MUST define how framework services, like the Conditional Permission Admin service, interact with the JVM singleton Security Manager.

6. The solution SHOULD specify a common multiplexing mechanism that can be shared by other singleton services.

7. The solution MUST provide a mechanism for parent frameworks to provide resources (e.g. packages and services, though potentially other resources as well) to child frameworks, including defining how state changes to resource providers in the parent framework impact resource consumers in the child framework.

8. The solution MAY provide a mechanism for peer frameworks to share resources (packages and services) with each other, including defining how state changes to resource providers one framework impact resource consumers in peer frameworks.

9. The solution MAY provide a mechanism for child frameworks to provide resources (packages and services) to its parent, including defining how state changes to resource providers in a child framework impact resource consumers in the parent framework.

10. The solution MUST enable mixed-vendor, multi-framework environments: a framework must be able to host, embed, or be a peer of another vendor's framework in the same JVM.

11. The solution MUST allow frameworks in the same JVM to be of different OSGi versions.

---

5 Technical Solution

The proposed technical solution preserves the following assumption: each framework manages its own class space, and is responsible for managing the lifecycle and resolution of its bundles. There is an implicit restriction of visibility based on a framework's classloading structure: class reachability inside the framework is managed by bundle resolution and classloading rules; class reachability outside the framework is managed by standard classloading rules and hierarchies, and doesn't have visibility into a framework's class space without the assistance of special bridging classloaders.
5.1 Multi-framework environments

5.1.1 Peer embedded frameworks

Use case: A non-OSGi runtime hosts two or more applications. Each application uses an OSGi framework as a localized (isolated) mini-runtime; function is delivered, managed, and updated using bundles hosted in that framework. The frameworks (A and B) are isolated from each other, and could be from different vendors.

Launching and using peer frameworks within a single JVM is covered by the mechanisms introduced in RFC 132, however, JVM singletons will require cross-framework multiplexing: framework services like the Thread IO service won't help, as the frameworks are disjoint.

- The lifecycle of each framework should be managed by the launching code.
- The parent classloader of each framework is the classloader used to instantiate the framework.

5.1.2 Nested frameworks

Use case: A bundle running within an OSGi framework uses a its own private framework instance to provide some function. This is essentially a variant of the peer embedded case.

Launching the nested framework is also covered by the mechanisms introduced in RFC 132. Due to the disjoint/isolated nature of the nested framework, cross-framework multiplexing is also necessary to address JVM singletons.

The nested framework (Framework C) is completely independent of (and hidden from) the hosting framework (Framework A): it is visible only to the launching bundle. As with peer embedded frameworks, the hosting and nested frameworks could be from different vendors.

- The lifecycle of the nested framework should be managed by the launching code (as with any embedded framework). In this case, it is the launcher's responsibility to ensure that the nested framework is stopped when the launching bundle is stopped.
- The parent classloader of the nested framework is the classloader of the launching bundle. Additional packages may have to be imported by the launching bundle (via Import-Package or DynamicImport-Package) in order to ensure that all packages from the VM execution environment are available to the nested framework classloader.

5.1.3 Parent/Child frameworks

Unlike plain embedded or nested frameworks, there is a direct and clearly defined relationship between the nested framework (the child), and the hosting framework (the parent).
A new OSGi service is defined to allow a framework to create another instance of itself: callers of the new API will be able to specify parameters (as they can for RFC 132) that influence how the new framework is created. In this environment, the parent and child frameworks are provided by the same vendor, and the parent framework is aware of its children.

To create the configuration shown in Figure 3, this service would be called by a bundle in Framework B to create Framework B1; code in a bundle in Framework B1 would use the service to create Framework B2. In this example, Framework B knows about B1 but not B2, only B1 is aware of B2. The lifecycle of each of these child frameworks is bound to the lifecycle of its parent: if Framework B is stopped, B1 is stopped, which in turn causes B2 to stop.

- The lifecycle of the child framework is bound to the lifecycle of the parent framework, not to the launching/starting bundle.
- The parent classloader of the child framework is the classloader of the parent framework's system bundle.

### 5.1.4 Identifying the properties used to start a framework

Given the potential for multiple frameworks to be present in the VM (for any of the scenarios above), the Framework should return the following properties used during framework creation when the `getHeaders` method is called:

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bundle-ContactAddress</td>
<td>Optional; recommended</td>
</tr>
<tr>
<td>Bundle-Copyright</td>
<td>Optional; recommended</td>
</tr>
<tr>
<td>Bundle-Description</td>
<td>Optional; recommended</td>
</tr>
<tr>
<td>Bundle-DocURL</td>
<td>Optional; recommended</td>
</tr>
<tr>
<td>Bundle-Icon</td>
<td>Optional; recommended</td>
</tr>
<tr>
<td>Bundle-Localization</td>
<td>Optional; recommended</td>
</tr>
<tr>
<td>Bundle-ManifestVersion</td>
<td>Mandatory: the highest manifest version this framework understands</td>
</tr>
<tr>
<td>Bundle-Name</td>
<td>Optional; recommended</td>
</tr>
<tr>
<td>Bundle-RequiredExecutionEnvironment</td>
<td>Mandatory: the list of execution environments supported by this framework</td>
</tr>
<tr>
<td>Bundle-SymbolicName</td>
<td>Mandatory: The specific name for the framework</td>
</tr>
<tr>
<td>Bundle-Vendor</td>
<td>Optional; recommended</td>
</tr>
<tr>
<td>Bundle-Version</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Export-Package</td>
<td>Contains org.osgi.framework.system.packages</td>
</tr>
</tbody>
</table>

The following headers (which are not applicable to frameworks) should not appear in the dictionary:

- Bundle-ActivationPolicy
- Bundle-Activator
- Bundle-Category
- Bundle-ClassPath
- Bundle-NativeCode
- Bundle-UpdateLocation
- DynamicImport-Package
- Export-Service
- Fragment-Host
5.2 Managing a Child Framework - Provisional

5.2.1 Composite and Surrogate Bundles

A child framework will be managed using a pair of Bundles: One that represents the child framework in the parent (the Composite bundle), and one that represents the parent framework in the child (the Surrogate bundle). OSGi framework providers are responsible for providing implementations of the Composite and Surrogate bundle interfaces.

- org.osgi.service.framework.CompositeBundle extends org.osgi.framework.Bundle
  - Framework getCompositeFramework() – returns the child framework associated with this bundle.
  - SurrogateBundle getSurrogateBundle() – returns the surrogate bundle.
  - update(Map compositeManifest) – Both the composite and surrogate bundles share some manifest information, like bundle symbolic name and version. The specified manifest can also include information about packages and services shared between the two frameworks. Any of the headers provided in the composite manifest can change when the bundle is updated.
  - void update() and void update(InputStream input) – These methods mark a divergence from regular bundles: the nature of composite bundles, and their relationship with a child framework dictate that standard update methods should not apply. It should not be possible to arbitrarily replace the content of a composite bundle, since the content of that bundle is defined by the manifest provided when the composite bundle and child framework are constructed (see below) or updated via the update(compositeManifest) method.
  - void uninstall() – this is a standard method on the Bundle interface. What is significant is that uninstalling a composite bundle will cause the associated child framework to be shutdown.

- org.osgi.service.framework.SurrogateBundle extends org.osgi.framework.Bundle
  - BundleContext getCompositeBundleContext() – returns the BundleContext for the composite bundle. The parent framework can be retrieved from the returned BundleContext, if necessary.
  - void update() and void update(InputStream input) – These methods mark a divergence from regular bundles: the nature of surrogate bundles, and their relationship with a child framework dictate that standard update methods should not apply. It should not be possible to arbitrarily replace the content of a surrogate bundle, since the content of that bundle is defined by the manifest provided when the composite bundle and child framework are constructed (see below) or updated via the update(compositeManifest) method on the associated composite bundle.
  - void uninstall() – this is a standard method on the Bundle interface that is not supported for surrogate bundles.
Aside from the exceptions mentioned in the brief method overviews above, composite and surrogate bundles interact with the hosting framework as any other bundle would: package resolution, package refresh operations, bundle start level semantics all apply to composite and surrogate bundles just as they would to any other bundle.

5.2.2 Creating a child framework

Frameworks must provide an instance of `org.osgi.service.framework.CompositeBundleFactory` to support the creation of child frameworks. The `CompositeBundleFactory` interface defines a method for creating a child framework and its associated composite and surrogate bundles:

- `CompositeBundle installCompositeBundle (Map frameworkConfig, String location, Map compositeManifest)`

  - This method does a few things (the details are in the Javadoc in section 7):
    - Creates a new composite bundle.
    - Creates and initializes a new child framework, using space associated with the persistent storage area of the composite bundle for the child framework's storage area (this implies that the `org.osgi.framework.storage` property, if provided in the framework configuration map, is ignored).
    - Creates a new surrogate bundle, and installs it into the child framework.
    - Updates the composite bundle's state to INSTALLED, and returns the child composite bundle.

  - The framework configuration Map defines properties that should be used to initialize the child framework, and must follow the guidelines specified in RFC 132. This parameter may be null, in which case it is up to the framework to define reasonable defaults. Framework implementations may also prevent certain properties from being defined for child frameworks, or may override provided properties with values required to satisfy their implementation (such properties should be clearly documented). This is the preferred mechanism for passing the equivalent of System properties to the child framework.

  - As noted above, the `org.osgi.framework.storage` property is ignored for the child framework, as the storage area is allocated by the parent framework and managed with the persistent storage area of the composite bundle.

  - The location parameter defines the bundle location used for both the composite and surrogate bundles.

  - The location string is checked when the `installCompositeBundle` method is called: if a bundle with the same location string is already installed, and it is a composite bundle, that bundle is returned; otherwise a `BundleException` is thrown to indicate that an incompatible bundle is already installed at the requested location.

  - The composite manifest Map defines:
    - Manifest headers: The composite and surrogate bundles associated with the new child framework will share the same `Bundle-SymbolicName` and `Bundle-Version`. The map provided as a parameter to `installCompositeBundle` must contain the `Bundle-SymbolicName` that the two link bundles will share. The `Bundle-Version` header is optional.
Shared Resources: If resources are to be shared between the two frameworks (in either direction), the map provided as a parameter to `installCompositeBundle` must contain entries describing the shared resources:

- **Import-Package** – the packages which are imported from the parent framework by the composite bundle, and are exported to the child framework by the surrogate bundle.

- **Export-Package** – the packages which are imported from the child framework by the surrogate bundle, and are exported to the parent framework by the composite bundle.

- **CompositeServiceFilter-Import** – a list of service filters that will be used by the composite bundle to acquire services from the parent framework; these services will then be registered in the child framework by the surrogate bundle.

- **CompositeServiceFilter-Export** – a list of service filters that will be used by the surrogate bundle to acquire services from the child framework; these services will then be registered in the parent framework by the composite bundle.

The manifest Map must not contain the following headers: `Bundle-ActivationPolicy`, `Bundle-Activator`, `Bundle-ClassPath`, `Bundle-Localization`, `Bundle-NativeCode`, `DynamicImport-Package`, or `Fragment-Host`. The `Bundle-ManifestVersion` header may be present, but its value must be at least 2. A `BundleException` will be thrown if invalid manifest headers are present. A framework implementation can decide whether or not `Require-Bundle` is supported, if not, the framework should throw a `BundleException` if the header is present in the manifest map.

As mentioned in 5.1.3, the lifecycle of frameworks created in this way is managed by the parent/owning framework. Because the child framework is associated with a composite bundle, the framework can manage the lifecycle of child frameworks using the standard mechanisms used for managing bundles.

Because of the association between the child framework's storage area and the composite bundle's persistent data area, child frameworks can be automatically restarted— the framework should have all of the information it needs to reify both the composite bundle, the child framework, and the surrogate bundle.

Composite and surrogate bundles look like any other bundles installed in the framework, but there are some restrictions:

- The standard update methods can not be used on composite or surrogate bundles: it should not be possible to upgrade the contents of a composite or surrogate bundle through any other means than providing updated manifest headers to the composite bundle.

- A framework may have multiple composite bundles installed, but only one surrogate bundle. For example, in Figure 5: Framework A is a parent framework, it has a composite bundle installed; Framework A1 is both a child and a parent framework, and so has both a composite bundle and a
surrogate bundle installed; and Framework A2, which is a child framework, has only a surrogate bundle installed.

When sharing services between parent and child frameworks, the composite and surrogate bundles track shared services based on the filters provided when they were created (or last updated). Changes in the availability of shared services are mirrored by registration/de-registration of services by those bundles: in this way, consumers of shared services in the consuming framework will be exposed to the dynamics of shared services in the providing framework just as they would be if the service was provided by a regular bundle in the same framework.

The resources shared between a parent and child framework via the composite and surrogate bundles can be dynamically updated by calling the `update(Map compositeManifest)` method on the composite bundle with a revised manifest map. Similar to normal bundle updates, the packages exported by a composite or surrogate bundle can not change as a result of calling update: the previous package exports must be available to other consuming bundles (in either the parent or child framework) until the `PackageAdmin.refreshPackages` method has been called to refresh the composite, or the parent Framework is relaunched.

More information about the behavior of these interfaces (including lifecycle and other interactions) are outlined in the attached Javadoc (Section 7).

### 5.3 Sharing packages and services with composite bundles - Provisional

As mentioned above, a composite bundle may be used to share packages and services between a parent and child framework.

#### 5.3.1 Importing packages

The Import-Package header is used to declare packages which are imported into a composite. The Import-Package header for a composite is allowed to specify any of the defined directives from the OSGi specification for the Import-Package header. Currently the only directive defined for Import-Package is the resolution directive. The Import-Package header for a composite is also allowed to specify any matching attributes that it requires. This includes the predefined matching attributes (e.g version, bundle-symbolic-name, bundle-version) as well as arbitrary matching attributes.

When resolving a composite bundle a wire is created for each import package constraint which the composite declares in the Import-Package header. These package wires are associated with an exported package from another bundle which is installed in the same framework as the composite bundle (the parent framework). The surrogate bundle, installed in the child framework, will then export the package. The package exported by the surrogate bundle represents the exported package from the parent framework. The package exported from the surrogate bundle must declare all the defined directives and attributes from the original exported package, which exists in the parent framework, with the following exceptions:

- bundle-symbolic-name - This attribute will use the symbolic name of the surrogate bundle. The symbolic name of the surrogate bundle is defined at composite installation and update time.
- bundle-version - This attribute will use the bundle version of the surrogate bundle. The bundle version of the surrogate bundle is defined at composite installation and update time.

Note that optional imports are allowed for composites (using the directive resolution:=optional). The surrogate bundle will export a package for an optional import only if the optional import for the composite is resolved to an exported package in the parent framework.
5.3.1.1 Packages from the system bundle

A composite bundle may import a package that ends up being wired to an exported package from the parent framework's system bundle. A common scenario where this can happen is when a composite is importing a package which uses another package that is exported by the system bundle. For example, imagine the following bundle is installed into the parent framework:

```
Bundle-SymbolicName: org.foo
Export-Package: org.foo; version="1.3"; uses="org.osgi.framework"
Import-Package: org.osgi.framework
```

This bundle exports the org.foo package which uses the org.osgi.framework package. It also imports the org.osgi.framework package. When resolving this bundle the import for org.osgi.framework package will be resolved to and exported package from the parent framework's system bundle.

When a composite bundle imports a package it should also import any packages that the original package uses. This is needed to ensure class spaced consistency in the child framework. In the example above, if a composite bundle imports the org.foo package then it should also import the org.osgi.framework package. This will result in the composite bundle be wired to the exported package org.osgi.framework from the parent framework's system bundle. This will result in a surrogate bundle that exports both the org.foo and org.osgi.framework package in the child framework. Now two exported packages in the child framework will exist for the same package org.osgi.framework (one from the child system bundle and one from the surrogate bundle). Both exported packages should end up using the same source class loader from the parent system bundle to load content from.

In this case, a second exported package in the child framework for the org.osgi.framework package should not be allowed. The framework must determine when a composite is wired to an exported package from the system bundle and it must not allow that package to be exported by the surrogate bundle. The Framework must still ensure that a consistent class space is preserved in the child framework.

One way to do this is to cause the surrogate bundle to export and import the system bundle package. The import would specify a matching attribute “bundle-symbolic-name=system.bundle” to ensure that the import is wired to the system bundle in the child framework. This would result in the export being dropped from the surrogate bundle at resolution time.

5.3.2 Exporting packages

The Export-Package header is used to declare packages which are exported from a composite. The Export-Package header for a composite is allowed to specify any of the supported directives which a normal bundle can specify. The Export-Package header for a composite bundle is also allowed to specify arbitrary matching attributes. Like normal bundles, a composite bundle may not specify the bundle-symbolic-name or bundle-version attributes on the Export-Package header.

A package exported from a composite bundle represents an exported package from another bundle which is installed in the child framework. During the composite bundle resolution process the exports from the composite must be validated. This is an additional resolution step required for composite bundles which is not required for normal bundles. For normal bundles, the framework does not verify that the exporting bundle actually contains the package which it specifies an export for.

When resolving a composite bundle, which exports packages, the surrogate bundle is used to import packages from other bundles installed in the child framework. Before allowing a composite bundle to resolve the framework must verify that the surrogate bundle's imported packages can be resolved. When resolving a surrogate bundle a wire is created for each import package constraint which the surrogate imports. These package wires are associated with an exported package from another bundle which is installed in the child framework. Before allowing the surrogate bundle to resolve each potential exported package wire must be verified against the export
package declarations of the composite bundle with the same package name. The exported package in the child
must have identical directives and attributes specified as the export package specification in the composite bundle
with the following exceptions:

- bundle-symbolic-name - This attribute will use the symbolic name of the composite bundle. The symbolic
  name of the composite bundle is defined at composite installation and update time.
- bundle-version - This attribute will use the bundle version of the composite bundle. The bundle version of
  the composite bundle is defined at composite installation and update time.

5.3.3 Composite resolution process

TBD. Much of this is outlined in the previous 2 sections but a step by step outline of the complete process for
resolving a composite bundle would be useful ...

5.3.4 Importing services

The CompositeServiceFilter-Import header is used to select services in the parent framework which will be
imported into the child framework. When a composite bundle is started the framework will use the the composite
bundle's context to listen for service references that match one or more of the filters specified by the
CompositeServiceFilter-Import header. If a service reference is found that matches one or more of the filters then
a corresponding service is registered in the child framework by using the surrogate bundle's context. When
registering the corresponding service with the surrogate bundle context the service properties of the original
service reference must be used to register the service with the following exceptions:

- service.id – The value of this property is assigned by the child framework when the service is registered.
  The child framework assigns a unique value that is larger than all previously assigned values since the
  child framework was started.
- objectClass – This property is set by the child framework when the service is registered. When
  registering the service with the surrogate bundle's context the value of the original objectClass service
  property must be used as the clazzes argument (i.e. BundleContext.registerService(String[] clazzes,
  Object service, Dictionary properties)).

5.3.5 Exporting services

The CompositeServiceFilter-Export header is used to select services in the child framework which will be exported
into the parent framework. When a surrogate bundle is started the framework will use the the surrogate bundle's
context to listen for service references that match one or more of the filters specified by the
CompositeServiceFilter-Export header. If a service reference is found that matches one or more of the filters then
a corresponding service is registered in the parent framework by using the composite bundle's context. When
registering the corresponding service with the composite bundle context the service properties of the original
service reference must be used to register the service with the following exceptions:

- service.id – The value of this property is assigned by the parent framework when the service is registered.
  The parent framework assigns a unique value that is larger than all previously assigned values since the
  parent framework was started.
- objectClass – This property is set by the parent framework when the service is registered. When
  registering the service with the composite bundle's context the value of the original objectClass service
  property must be used as the clazzes argument (i.e. BundleContext.registerService(String[] clazzes,
  Object service, Dictionary properties)).
5.4 Singleton Multiplexing

5.4.1 Multiplexing factories - Provisional

An interface defined as part of the OSGi specification will allow multi-vendor, multi-framework environments to correctly support the URL Handlers Service Specification without conflicting. The following is based on an algorithm outlined by Simon Kaegi [3] and the implementation used by Equinox.

- `org.osgi.multiplexer.MultiplexSupport`: a simple API that framework providers can use (via reflection) to discover the presence of other multiplexers.

  - `boolean supportsOSGiMultiplexing()` – this method allows a framework to test a pre-registered JVM singleton to see if a) the singleton is aware of the OSGi multiplexing algorithm (the method is present), and if so, b) if it can support multiplexing in the current environment. Singleton multiplexing is heavily dependent on potentially JVM-specific reflection, so it is possible that a framework could find itself in an environment in which it can't support multiplexing, in which case, the implementation of this method would return false.

  - `Object getLegacySingleton()` – this method preserves the value of a pre-existing non-multiplexing JVM singleton. As new multiplexers are added, `getLegacySingleton()` should be called on the current (multiplexing) singleton, and the returned value (if not null), saved for later use.

- `Object getNextMultiplexer()`, `Object getPrevMultiplexer()`, `void setNextMultiplexer(Object multiplexer)`, `void setPrevMultiplexer(Object multiplexer)` – these methods are used to create and maintain a double-linked list of registered multiplexers. This list is used both for insertion and removal of multiplexers as the associated frameworks startup and shutdown, but is also used for delegation between multiplexers to handle object operations.

- Registration of multiplexers: A framework will create an object (x) that satisfies the MultiplexSupport interface, and will first check to see if a JVM singleton is already registered.

  - If a non-multiplexer singleton exists (s), x will store that singleton, and will then use reflection to establish itself as the JVM singleton in place of s. Note that there may be other cached data associated with the previous singleton that must also be cleared when the singleton is reset.

  - If a multiplexer already exists (y), x will call `y.getLegacySingleton()` to preserve delegation to the non-multiplexing singleton s. x must then use `y.getNextMultiplexer()` to add itself to the chain of multiplexers. Frameworks may search the chain for an implementation of “their” multiplexer (such that all multiplexers for a certain frameworks share one link in the chain), or may simply append additional multiplexer instances to the chain. In either case, it is important to maintain the double-linked nature of the list when it is necessary to add themselves to the list (`n.setNextMultiplexer(x)`, and `x.setPrevMultiplexer(n)`).

- De-registration of multiplexers: a framework will find its multiplexer in the chain (how that happens may vary, depending on how the framework adds multiplexers to the chain):

  - If the multiplexer is the set as the JVM singleton, it will use reflection to re-set the JVM singleton to the next multiplexer in the chain, or to the legacy singleton. If a multiplexer is selected, it's “previous” link should be cleared, to indicate that it is the new head of the chain.
○ If the multiplexer is in the middle or end of the chain, it should make the appropriate method calls to reset the previous/next links of neighboring multiplexers (n.setNextMultiplexer(m), and m.setPrevMultiplexer(n)) in order to preserve the double-linked list.

● Looking up the right multiplexer: the JVM singleton multiplexer should first check to see (via implementation specific means) if it should handle the request. Each framework implementation will have its own way of determining the caller context (and hence determining whether or not “this” multiplexer should handle the method call). If the multiplexer determines that it needs to delegate the method call, it will first call the requisite method on the “next” multiplexer in the chain, if present. If there isn’t a “next” multiplexer, the method call should be delegated to the legacy singleton.

5.4.2 Other VM Singletons

● System Properties – System properties are an unavoidable JVM singleton. The new framework launching mechanism defined by RFC 132 allows properties to be provided at framework creation time that may or not be backed by System properties.

● System.in, System.out, System.err – see Thread IO service defined in RFC 132 for a potential solution to these JVM singletons.

5.4.3 OSGi Singletons

● org.osgi.framework.FrameworkUtil is an existing class that defines a single static method used to retrieve a Filter object for looking up Service References or Dictionary objects. Framework implementations override this class in order to provide their own Filter implementations.

○ The static FrameworkUtil.createFilter method was modified to return a common Filter implementation. Framework implementations are not required provide their own implementations of FrameworkUtil. The BundleContext.createFilter method can be used to create a Filter that is optimized for particular framework implementations.

● org.osgi.framework.AdminPermission and org.osgi.service.condpermadmin.BundleSignerCondition both end up tied to particular framework implementations because there is no generic way to get the signer DN chains for a bundle.

○ A new method, boolean matchDistinguishedNameChain(String, List) has been added to org.osgi.framework.FrameworkUtil to provide a common DN chain matching algorithm.

○ A new method, Map getSignerCertificates(int), has been added to org.osgi.framework.Bundle to facilitate retrieval of the DN chains used to sign a Bundle.

○ Final class org.osgi.framework.AdminPermission now uses these two methods to work with the DN chain for a bundle.

○ org.osgi.service.condpermadmin.BundleSignerCondition has also been updated to use the new methods.

5.5 Open Issues

Other Singleton resource providers?

● javax.xml.parsers.SAXParserFactory / javax.xml.parsers.DocumentBuilderFactory
Security Manager – there is no apparent way to multiplex the JVM Singleton security manager, nor are the implications of doing such a thing obvious. With multiple frameworks present in the same VM, the first to establish a SecurityManager effectively “wins”: the VM will call that instance of the security manager for all subsequent permission checks. In the case of parent/child frameworks of the same type (e.g. a hierarchy of Equinox frameworks), the framework implementation is free to perform additional checks to support different rules/permissions defined for each framework instance in the hierarchy. How such interactions might take place between disjoint frameworks (especially if the framework implementations differ) is undefined.

6 Security Considerations

There are several open issues regarding security considerations for using multiple frameworks in the same JVM:

- The security manager delegates to the protection domain of the individual bundles when performing permission checks. Because every bundle has its own protection domain, the presence of multiple frameworks should not impact permission checks in general.

- The presence/use of nested frameworks may impact the behavior of postponed conditions in the ConditionalPermissionAdmin service. More investigation and clarification is needed in this area.

- A check for AllPermission will be performed when creating a new nested framework if a SecurityManager is enabled. Per RFC 132, AllPermission is needed to create a framework—this requirement extends to the creation of nested or child frameworks as well. A framework implementation must ensure that the composite bundles it creates have sufficient permissions to do the necessary import/export/service tracking operations.
7 Javadoc

7.1 Interface org.osgi.framework.Bundle

All Known Subinterfaces:
    CompositeBundle, Framework

public interface Bundle

An installed bundle in the Framework.

A Bundle object is the access point to define the lifecycle of an installed bundle. Each bundle installed in the OSGi environment must have an associated Bundle object.

A bundle must have a unique identity, a long, chosen by the Framework. This identity must not change during the lifecycle of a bundle, even when the bundle is updated. Uninstalling and then reinstalling the bundle must create a new unique identity.

A bundle can be in one of six states:
- UNINSTALLED
- INSTALLED
- RESOLVED
- STARTING
- STOPPING
- ACTIVE

Values assigned to these states have no specified ordering; they represent bit values that may be OR-ed together to determine if a bundle is in one of the valid states.

A bundle should only execute code when its state is one of STARTING, ACTIVE, or STOPPING. An UNINSTALLED bundle can not be set to another state; it is a zombie and can only be reached because references are kept somewhere.

The Framework is the only entity that is allowed to create Bundle objects, and these objects are only valid within the Framework that created them.

ThreadSafe

Field Summary
static int SIGNERS_ALL
Request that all certificates used to sign the bundle be returned.

static int SIGNERS_TRUSTED
Request that only certificates used to sign the bundle that are trusted by the framework be returned.

## Method Summary

| java.util.Map | getSignerCertificates(int signersType) | Return the certificates for the signers of this bundle and the certificate chains for those signers. |

## Field Detail

### 7.1.1 SIGNERS_ALL

```java
static final int SIGNERS_ALL
```
Request that all certificates used to sign the bundle be returned.

**Since:**
1.5

**See Also:**
- `getSignerCertificates(int)`, [Constant Field Values](#)

### 7.1.2 SIGNERS_TRUSTED

```java
static final int SIGNERS_TRUSTED
```
Request that only certificates used to sign the bundle that are trusted by the framework be returned.

**Since:**
1.5

**See Also:**
- `getSignerCertificates(int)`, [Constant Field Values](#)

## Method Detail

### 7.1.3 getSignerCertificates

```java
java.util.Map getSignerCertificates(int signersType)
```
Return the certificates for the signers of this bundle and the certificate chains for those signers.

**Parameters:**
- `signersType` - If SIGNERS_ALL is specified, then information on all signers of this bundle is returned. If SIGNERS_TRUSTED is specified, then only information on the signers of this bundle trusted by the framework is returned.

**Returns:**
The X509Certificates for the signers of this bundle and the X509Certificate chains for those signers. The keys of the Map are the X509Certificates of the signers of this bundle. The value for a key is a List containing the X509Certificate chain for the signer. The first item in the List is the signer's X509Certificate which is then followed by the rest of the X509Certificate chain. The returned Map will be empty if there are no signers.

Throws:
java.lang.IllegalArgumentException - If the specified signersType is not SIGNERS_ALL or SIGNERS_TRUSTED.

Since:
1.5
7.2 Class org.osgi.framework.FrameworkUtil

public class FrameworkUtil extends java.lang.Object

Framework Utility class.

This class contains utility methods which access Framework functions that may be useful to bundles.

Since: 1.3
ThreadSafe

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Filter createFilter(java.lang.String filter)</td>
<td>Creates a Filter object. This Filter object may be used to match a ServiceReference object or a Dictionary object. If the filter cannot be parsed, an InvalidSyntaxException will be thrown with a human readable message where the filter became unparsable. This method returns a Filter implementation which may not perform as well as the framework implementation-specific Filter implementation returned by BundleContext.createFilter(String).</td>
</tr>
<tr>
<td>static boolean matchDistinguishedNameChain(java.lang.String matchPattern, java.util.List dnChain)</td>
<td>Matches a Distinguished Name (DN) chain against a pattern.</td>
</tr>
</tbody>
</table>

Methods inherited from class java.lang.Object

close, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

Method Detail

7.2.1 createFilter

public static Filter createFilter(java.lang.String filter) throws InvalidSyntaxException

Creates a Filter object. This Filter object may be used to match a ServiceReference object or a Dictionary object.

If the filter cannot be parsed, an InvalidSyntaxException will be thrown with a human readable message where the filter became unparsable.

This method returns a Filter implementation which may not perform as well as the framework implementation-specific Filter implementation returned by BundleContext.createFilter(String).

Parameters:
filter - The filter string.

Returns:
A Filter object encapsulating the filter string.

Throws:
- InvalidSyntaxException - If filter contains an invalid filter string that cannot be parsed.
- java.lang.NullPointerException - If filter is null.

See Also:
Filter

7.2.2 matchDistinguishedNameChain

public static boolean matchDistinguishedNameChain(java.lang.String matchPattern,
java.util.List<java.lang.String> dnChain)

Matches a Distinguished Name (DN) chain against a pattern. DNs can be matched using wildcards. A wildcard ("*" *) replaces all possible values. Due to the structure of the DN, the comparison is more complicated than string-based wildcard matching.

A wildcard can stand for zero or more DNs in a chain, a number of relative distinguished names (RDNs) within a DN, or the value of a single RDN. The DNs in the chain and the matching pattern are canonicalized before processing. This means, among other things, that spaces must be ignored, except in values.

The format of a wildcard match pattern is:

matchPattern ::= dn-match ( ';' dn-match ) *

- dn-match ::= ( '*' | rdn-match ) ( ',' rdn-match ) * | '-'
- rdn-match ::= name '=' value-match
- value-match ::= '*' | value-star
- value-star ::= < value, requires escaped '*' and '-' >

The most simple case is a single wild-card; it must match any DN. A wild-card can also replace the first list of RDNs of a DN. The first RDNs are the least significant. Such lists of matched RDNs can be empty.

For example, a match pattern with a wild-card that matches all all DNs that end with RDNs of o=ACME and c=US would look like this:

*, o=ACME, c=US

This match pattern would match the following DNs:

- cn = Bugs Bunny, o = ACME, c = US
- ou = Carrots, cn=Daffy Duck, o=ACME, c=US
- street = 9C\ Avenue St. DrA@zA@ry, o=ACME, c=US
- dc=www, dc=acme, dc=com, o=ACME, c=US
- o=ACME, c=US

The following DNs would not match:
If a wildcard is used for a value of an RDN, the value must be exactly `*`. The wild-card must match any value, and no substring matching must be done. For example:

```
cn=*,o=ACME,c=*  
```

This match pattern with wild-card must match the following DNs:

```
cn=Bugs Bunny,o=ACME,c=US  
cn=Daffy Duck,o=ACME,c=US  
cn=Road Runner,o=ACME,c=NL  
```

But not:

```
o=ACME,c=NL  
dc=acme.com,cn=Bugs Bunny,o=ACME,c=US  
```

A match pattern may contain a chain of DN match patterns. The semicolon(`;`) must be used to separate DN match patterns in a chain. Wild-cards can also be used to match against a complete DN within a chain.

The following example matches a certificate signed by Tweety Inc. in the US.

```
*; ou=S & V, o=Tweety Inc., c=US  
```

The wild-card (`*`) matches zero or one DN in the chain, however, sometimes it is necessary to match a longer chain. The minus sign (`-`) represents zero or more DNs, whereas the asterisk only represents a single DN. For example, to match a DN where the Tweety Inc. is in the DN chain, use the following expression:

```
-; *, o=Tweety Inc., c=US  
```

Parameters:
- `dnChain` - A DN chain. Each element of the chain must be of type String and use the format defined in RFC 2253.
- `matchPattern` - a pattern to match the DN chain against

Returns:
- true of the pattern matches the DN chain; otherwise false is returned

Throws:
- `java.lang.IllegalArgumentException` - if the match pattern or the DN chain is invalid.
7.3 Interface org.osgi.multiplexer.MultiplexSupport - Provisional

public interface MultiplexSupport

This is a simple API intended to allow multiple frameworks in the same JVM to "share" JVM singletons like the URLStreamHandlerFactory and the ContentHandlerFactory. Supporting this API makes heavy use of potentially JVM-specific reflection to manage both the JVM singleton instance and any cached state associated with previously set singletons.

Since each multiplexer could be associated with classes loaded by a different classloader, the methods on this interface would have to be discovered and invoked via reflection, rather than by casting or instanceof tests.

The presence and support of this interface (as determined by the supportsOSGiMultiplexing method) implies support of the following multiplexing algorithm for a given JVM singleton.

7.3.1.1 Initialization:

- Create a new multiplexer for the given JVM singleton, e.g. an implementation of a URLStreamHandlerFactory that also implements the MultiplexSupport interface.
- Retrieve the current JVM Singleton-- Does it support multiplexing?
  - If yes, get (and store) the legacy singleton from the JVM singleton multiplexer in the new multiplexer, and append the new multiplexer to the double-linked list.
  - If no, store the current singleton in the new multiplexer as the "Legacy singleton", and use Java reflection to reset the JVM singleton instance and clear any cached state before setting the new multiplexer as the JVM singleton.

7.3.1.2 Termination:

- When a framework is cleaning up, it will need to "remove" its multiplexer from the double-linked list, in a process that is the reverse of what happens during initialization.
  - If the multiplexer being cleaned up is registered as the JVM singleton, then it needs to set a new singleton: either the "next" multiplexer in the chain if other multiplexers exist, or the legacy singleton, if it exists.
  - If the multiplexer being cleaned up is not the JVM singleton, it needs only to update the previous and next links of its neighbors in order to remove itself from the list.

7.3.1.3 Determining the context:

When a method is called on a multiplexing JVM singleton, the singleton (and any subsequently called multiplexers) must decide whether or not they are the correct instance for the method call. If the current multiplexer determines that it cannot handle the current method call, it should delegate to the next multiplexer in the chain. The last multiplexer in the chain should delegate method calls to the legacy singleton if the calling context doesn't match.
## Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.Object getLegacySingleton()</code></td>
<td>Obtain a reference to the pre-existing non-multiplexing JVM singleton, if present.</td>
</tr>
<tr>
<td><code>java.lang.Object getNextMultiplexer()</code></td>
<td></td>
</tr>
<tr>
<td><code>java.lang.Object getPrevMultiplexer()</code></td>
<td></td>
</tr>
<tr>
<td><code>void setNextMultiplexer(java.lang.Object multiplexer)</code></td>
<td>Set the next multiplexer in the chain.</td>
</tr>
<tr>
<td><code>void setPrevMultiplexer(java.lang.Object multiplexer)</code></td>
<td>Set the previous multiplexer in the chain.</td>
</tr>
<tr>
<td><code>boolean supportsOSGiMultiplexing()</code></td>
<td>This method allows a framework to test a pre-registered JVM singleton to see if a) the singleton is aware of the OSGi multiplexing algorithm (the method is present), and if so, b) if it can support multiplexing in the current environment.</td>
</tr>
</tbody>
</table>

## Method Detail

### 7.3.2 supportsOSGiMultiplexing

```java
boolean supportsOSGiMultiplexing()
```

This method allows a framework to test a pre-registered JVM singleton to see if a) the singleton is aware of the OSGi multiplexing algorithm (the method is present), and if so, b) if it can support multiplexing in the current environment.

Singleton multiplexing is heavily dependent on potentially JVM-specific reflection, so it is possible that a framework could find itself in an environment in which it can't support multiplexing, in which case, the implementation of this method would return false.

**Returns:**
true if multiplexing supported in the current JVM, false otherwise.

### 7.3.3 getLegacySingleton

```java
java.lang.Object getLegacySingleton()
```

Obtain a reference to the pre-existing non-multiplexing JVM singleton, if present. As new multiplexers are added, getLegacySingleton() should be called on the current (multiplexing) singleton, and the returned value (if not null), should be saved and used for delegation.

### 7.3.4 getNextMultiplexer

```java
java.lang.Object getNextMultiplexer()
```
7.3.5 setNextMultiplexer

```java
void setNextMultiplexer(java.lang.Object multiplexer)
```

Set the next multiplexer in the chain.

**Parameters:**
- multiplexer - The next multiplexer in the double-linked list.

7.3.6 getPrevMultiplexer

```java
java.lang.Object getPrevMultiplexer()
```

**Returns:**
- the previous multiplexer in the chain, or null if not set.

7.3.7 setPrevMultiplexer

```java
void setPrevMultiplexer(java.lang.Object multiplexer)
```

Set the previous multiplexer in the chain.

**Parameters:**
- multiplexer - The previous multiplexer in the double-linked list.
7.4 Interface org.osgi.service.framework.CompositeBundle - Provisional

All Superinterfaces:
Bundle

public interface CompositeBundle extends Bundle

Composite bundles are composed of other bundles. The component bundles which make up the content of a composite bundle are installed into a child framework. Like a normal bundle, a composite bundle may import packages and use services from other bundles which are installed in the same framework as the composite bundle. The packages imported and the services used by a composite bundle are shared with the components of a composite bundle through a surrogate bundle installed in the child framework. Also like a normal bundle, a composite bundle may export packages and register services which can be used by bundles installed in the same framework as the composite bundle. The packages exported and the services registered by a composite bundle are acquired from the components of a composite bundle by the surrogate bundle installed in the child framework.

A framework has one composite bundle for each of its child frameworks. A framework can have zero or more composite bundles installed. A child framework must have one and only one surrogate bundle which represents the composite bundle in the parent framework. In other words, a parent framework can have many child frameworks but a child framework can have only one parent.

A composite bundle does the following as specified by the composite manifest map:
- Exports packages to the parent framework from the child framework. These packages are imported by the surrogate bundle installed in the child framework.
- Imports packages from the parent framework. These packages are exported by the surrogate bundle installed in the child framework.
- Registers services to the parent framework from the child framework. These services are acquired by the surrogate bundle installed in the child framework.
- Acquires services from the parent framework. These services are registered by the surrogate bundle installed in the child framework.

A newly created child Framework will be in the STARTING state. This child Framework can then be used to manage and control the child framework instance. The child framework instance is persistent and uses a storage area associated with the installed composite bundle. The child framework's lifecycle is tied to its composite bundle's lifecycle in the following ways:
- If the composite bundle is marked to be persistently started (see StartLevel.isBundlePersistentlyStarted(Bundle)) then the child framework instance will automatically be started when the composite bundle's start-level is met.
- The child framework instance will be stopped if the composite bundle is persistently stopped or its start level is no longer met. Performing operations which transiently stop a composite bundle do not cause the child framework to stop (e.g. stop(Bundle.STOP_TRANSIENT), update, refreshPackages etc.).
- If the composite bundle is uninstalled, the child framework's persistent storage area is also uninstalled.

The child framework may be persistently started and stopped by persistently starting and stopping the composite bundle, but it is still possible to initialize and start the child framework explicitly while the composite bundle is not persistently started. This allows for the child framework to be initialized and populated with a set of bundles before starting the composite bundle. The set of bundles installed into the child framework are the component bundles which comprise the composite bundle.

If the child framework's lifecycle is also tied to the lifecycle of its parent framework. When the parent Framework enters the STOPPING state, all active child frameworks of that parent are shutdown using the Framework.stop() method. The parent framework must not enter the Bundle.RESOLVED state until all the child frameworks have completed their shutdown process. Just as with other Bundles, references to child...
frameworks (or the associated composite and surrogate bundles) become invalid after the parent framework has completed the shutdown process, and must not be allowed to re-initialize or re-start the child framework.

See Also:
SurrogateBundle
ThreadSafe

Field Summary

Fields inherited from interface org.osgi.framework.Bundle
ACTIVE, INSTALLED, RESOLVED, SIGNERS_ALL, SIGNERS_TRUSTED, START_ACTIVATION_POLICY,
START_TRANSIENT, STARTING, STOP_TRANSIENT, STOPPING, UNINSTALLED

Method Summary

Framework
getCompositeFramework()
Returns the child framework associated with this composite bundle.

SurrogateBundle
getSurrogateBundle()
Returns the surrogate bundle associated with this composite bundle.

void
uninstall()
Uninstalls this composite bundle.

void
update()
This operation is not supported for composite bundles.

void
update(java.io.InputStream input)
This operation is not supported for composite bundles.

void
update(java.util.Map compositeManifest)
Updates this composite bundle with the specified manifest.

Methods inherited from interface org.osgi.framework.Bundle
findEntries, getBundleContext, getBundleId, getEntry, getEntryPaths, getHeaders,
ger Headers, getLastModified, getLocation, getRegisteredServices, getResource,
ger Resources, getServicesInUse, getSignerCertificates, getState, getSymbolicName,
hasPermission, loadClass, start, start, stop, stop

Method Detail

7.4.1 getCompositeFramework

Framework getCompositeFramework()

Returns the child framework associated with this composite bundle.

Returns:
the child framework.
7.4.2 getSurrogateBundle

SurrogateBundle getSurrogateBundle()

Returns the surrogate bundle associated with this composite bundle. The surrogate bundle is installed in the child framework.

Returns: the surrogate bundle.

7.4.3 update

void update(java.util.Map compositeManifest) throws BundleException

Updates this composite bundle with the specified manifest.

Similar to normal bundle updates, the packages exported by a composite or surrogate bundle cannot change as a result of calling update: the previous package exports must be available to other consuming bundles (in either the parent or child framework) until the PackageAdmin.refreshPackages method has been called to refresh the composite, or the parent Framework is re-launched.

Parameters: compositeManifest - the new composite manifest.

Throws: BundleException - If the update fails.

See Also: CompositeBundleFactory.installCompositeBundle(Map, String, Map)

7.4.4 update

void update() throws BundleException

This operation is not supported for composite bundles. A BundleException of type invalid operation must be thrown.

Specified by: update in interface Bundle

Throws: BundleException - If the update fails.

See Also: Bundle.stop(), Bundle.start()
This operation is not supported for composite bundles. A BundleException of type invalid operation must be thrown.

**Specified by:**
update in interface Bundle

**Parameters:**
input - The InputStream from which to read the new bundle.

**Throws:**
BundleException - If the provided stream cannot be read or the update fails.

**See Also:**
Bundle.update()

### 7.4.6 uninstall

```java
void uninstall()
  throws BundleException
```

Uninstalls this composite bundle. The associated child framework is shutdown, and its persistent storage area is deleted.

**Specified by:**
uninstall in interface Bundle

**Throws:**
BundleException - If the uninstall failed. This can occur if another thread is attempting to change this bundle's state and does not complete in a timely manner.

**See Also:**
Bundle.stop()
7.5 Interface org.osgi.service.framework.SurrogateBundle - Provisional

All Superinterfaces:
Bundle

public interface SurrogateBundle extends Bundle

A surrogate bundle is installed in a child framework, and is the child-facing representation of the composite bundle in the parent framework.

A surrogate bundle does the following as specified by the composite manifest:
- Exports packages to the child framework from the parent framework. These packages are imported by the composite bundle installed in the parent framework.
- Imports packages from the child framework. These packages are exported by the composite bundle installed in the parent framework.
- Registers services from the parent framework with the child framework. These services are acquired by the composite bundle installed in the parent framework.
- Acquires services from the child framework. These services are registered by the composite bundle installed in the parent framework.

See Also:
CompositeBundle

Field Summary

Fields inherited from interface org.osgi.framework.Bundle
ACTIVE, INSTALLED, RESOLVED, SIGNERS_ALL, SIGNERS_TRUSTED, START_ACTIVATION_POLICY, START_TRANSIENT, STARTING, STOP_TRANSIENT, STOPPING, UNINSTALLED

Method Summary

<table>
<thead>
<tr>
<th>BundleContext</th>
<th>getCompositeBundleContext()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the bundle context of the associated composite bundle.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>uninstall()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This operation is not supported for surrogate bundles.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>update()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This operation is not supported for surrogate bundles.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>update(java.io.InputStream input)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This operation is not supported for surrogate bundles.</td>
</tr>
</tbody>
</table>

Methods inherited from interface org.osgi.framework.Bundle
findEntries, getBundleContext, getBundleId, getEntry, getEntryPaths, getHeaders, getHeaders, getLastModified, getLocation, getRegisteredServices, getResource, getResources, getServicesInUse, getSignerCertificates, getState, getSymbolicName, hasPermission, loadClass, start, start, stop, stop
Method Detail

7.5.1 getCompositeBundleContext

BundleContext getCompositeBundleContext()

Returns the bundle context of the associated composite bundle.

Returns:
the bundle context of the composite bundle. A value of null is returned if the composite bundle
does not have a valid bundle context.

7.5.2 update

void update()

throws BundleException

This operation is not supported for surrogate bundles. A BundleException of type invalid
operation must be thrown.

Specified by:
update in interface Bundle

Throws:
BundleException - If the update fails.

See Also:
Bundle.stop(), Bundle.start()

7.5.3 update

void update(java.io.InputStream input)

throws BundleException

This operation is not supported for surrogate bundles. A BundleException of type invalid
operation must be thrown.

Specified by:
update in interface Bundle

Parameters:
input - The InputStream from which to read the new bundle.

Throws:
BundleException - If the provided stream cannot be read or the update fails.

See Also:
Bundle.update()

7.5.4 uninstall

void uninstall()

throws BundleException
This operation is not supported for surrogate bundles. A BundleException of type invalid operation must be thrown.

Specified by:
uninstall in interface Bundle

Throws:
BundleException - If the uninstall failed. This can occur if another thread is attempting to change this bundle's state and does not complete in a timely manner.

See Also:
Bundle.stop()
7.6 Interface org.osgi.service.framework.CompositeBundleFactory -
Provisional

public interface CompositeBundleFactory

Framework service that is used to create composite bundles.

If present, there will only be a single instance of this service registered with the Framework.

ThreadSafe

Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static java.lang.String COMPOSITE_SERVICE_FILTER_IMPORT</td>
<td>Manifest header (named &quot;CompositeServiceFilter-Import&quot;) identifying the service filters that are used by a composite bundle to select services that will be registered into a child framework by its associated surrogate bundle.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>installCompositeBundle</td>
<td>Installs a CompositeBundle.</td>
</tr>
</tbody>
</table>

Field Detail

7.6.1 COMPOSITE_SERVICE_FILTER_IMPORT

static final java.lang.String COMPOSITE_SERVICE_FILTER_IMPORT

Manifest header (named "CompositeServiceFilter-Import") identifying the service filters that are used by a composite bundle to select services that will be registered into a child framework by its associated surrogate bundle.

See Also:
Constant Field Values

7.6.2 COMPOSITE_SERVICE_FILTER_EXPORT

static final java.lang.String COMPOSITE_SERVICE_FILTER_EXPORT

Copyright © IBM Corporation 2009    All Rights Reserved
Manifest header (named "CompositeServiceFilter-Export") identifying the service filters that are used by a surrogate bundle to select services that will be registered into a parent framework by its associated composite bundle.

See Also:
Constant Field Values

Method Detail

7.6.3 installCompositeBundle

CompositeBundle installCompositeBundle(java.util.Map frameworkConfig,
java.lang.String location,
java.util.Map compositeManifest)
throws BundleException

Installs a CompositeBundle. The composite bundle has a new child Framework associated with it and a surrogate bundle which is installed in the child framework. Composite bundles share packages and services between the parent framework they are installed in and the child framework.

The following steps are required to create a composite bundle:
1. If a bundle containing the same location string is already installed and the Bundle object is a CompositeBundle, then that composite bundle is returned; otherwise a BundleException is thrown indicating that an incompatible bundle is already installed at the specified location.
2. The composite bundle's associated resources are allocated. The associated resources minimally consist of a unique identifier and a persistent storage area. If this step fails, a BundleException is thrown.
3. The compositeManifest map is used to provide the headers for the composite bundle and its surrogate bundle.

If composite manifest map does not contain the following header(s) then a BundleException is thrown:
- Bundle-SymbolicName the symbolic name used for the composite bundle and its surrogate bundle.

The composite manifest map may optionally contain the following header(s):
- Bundle-Version the bundle version used for the composite bundle and its surrogate bundle.
- Import-Package the packages which are imported from the parent framework by the composite bundle and are exported to the child framework by the surrogate bundle.
- Export-Package the packages which are imported from the child framework by the surrogate bundle and are exported to the parent framework by the composite bundle.
- CompositeServiceFilter-Import the service filters which are acquired from the parent framework by the composite bundle and are registered in the child framework by the surrogate bundle.
- CompositeServiceFilter-Export the service filters which are acquired from the child framework by the surrogate bundle and are registered in the parent framework by the composite bundle.
- Bundle-ManifestVersion the bundle manifest version. If this header is not specified then the default is to use version 2. A BundleException is thrown if this header is specified and the version is less than 2.
- Require-Bundle a bundle from the parent that is required by the child framework. Support for this header is experimental: a BundleException should be thrown if the header is present and the framework doesn't support it.
The composite manifest map must not contain the following headers. If a composite manifest map does contain one of the following headers then a BundleException is thrown:

- Bundle-ActivationPolicy
- Bundle-Activator
- Bundle-ClassPath
- Bundle-Localization
- Bundle-NativeCode
- Fragment-Host
- DynamicImport-Package

4. A child framework is created which uses a storage area associated with the composite bundle's persistent storage. The framework configuration property org.osgi.framework.storage, if specified, is ignored.

5. The child framework is initialized (see Framework.init()).

6. A surrogate bundle is created and installed into the child framework.

7. The composite bundle's state is set to INSTALLED.

8. A bundle event of type BundleEvent.INSTALLED is fired for the composite bundle.

9. The CompositeBundle object for the new composite bundle is returned.

Parameters:

- frameworkConfig - A map containing configuration parameters used to initialize and launch the child framework.
- location - The bundle location used for the composite and surrogate bundles.
- compositeManifest - A map containing the manifest used to create the composite and surrogate bundles

Returns:

A new composite bundle in INSTALLED state.

Throws:

- BundleException - If the composite manifest is invalid or there is some other problem with installing the composite bundle.
- java.lang.SecurityException - If the caller does not have AllPermission.

See Also:

- Framework, CompositeBundle

---

8 Considered Alternatives

This RFC has considered several alternatives related to child frameworks:

A FrameworkFactory service with two methods: newChildFramework, for creating a child framework, and newFrameworkLink, for defining a one-way FrameworkLink used to share resources between frameworks. The FrameworkLink was one directional, and used/managed two bundles to represent itself (and its shared resources) in the two frameworks. This idea included an immutable object called a LinkDescription, which contained the declarations of packages and services that would be shared by the link.
9 Document Support

9.1 References


9.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Erin Schnabel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>2455 South Road, Poughkeepsie, NY 12603</td>
</tr>
<tr>
<td>Voice</td>
<td>+1 845-435-5648</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:schnabel@us.ibm.com">schnabel@us.ibm.com</a></td>
</tr>
</tbody>
</table>

9.3 Acronyms and Abbreviations

9.4 End of Document
Abstract

This specification describes how Enterprise Web Applications written to Java Servlets and JSP specifications will be supported in OSGi. Such applications typically require services of a Web Container. The OSGi based Web Container supersedes the functionality of the OSGi Http Service, and can be implemented in a manner that supports bundles written to the original Http Service coexisting with bundles written to the new OSGi Web Container. The specification describes a model for Web application components to access OSGi services. It also describes how a OSGi based Web Container facilitates interoperability between Servlet and JSP application components with components written to other component models such as OSGi Declarative Services and Blueprint services (RFC 124).

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

Copyright © IBM Corporation 2003, 2009.
This contribution is made to the OSGi Alliance as MEMBER LICENSED MATERIALS pursuant to the terms of the OSGi membership agreement and specifically the license rights and warranty disclaimers as set forth in Sections 3.2 and 12.1, respectively.
All company, brand and product names contained within this document may be trademarks that are the sole property of the respective owners.
The above notice must be included on all copies of this document that are made.
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
  2.1.1 Structure of a Web application .......................................................................................5
  2.1.2 Web Application deployment descriptor (web.xml) .......................................................6
  2.1.3 Web application context root .......................................................................................6
  2.1.4 Temporary work area ....................................................................................................6
  2.1.5 JavaServer pages .........................................................................................................7

3 Problem Description ...................................................................................................................7

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

5 Technical Solution .......................................................................................................................8
  5.1 Architectural overview .......................................................................................................8
  5.2 Web application life cycle ..................................................................................................9
    5.2.1 Installing a Web application .......................................................................................9
    5.2.2 Starting a web application .......................................................................................11
    5.2.3 Stopping web application .......................................................................................12
    5.2.4 Uninstalling web application ..................................................................................12
  5.3 Accessing OSGi environment from Web application ........................................................12
  5.4 JavaServer Pages support ...............................................................................................13
    5.4.1 In-line compiling .....................................................................................................13
    5.4.2 Dynamic recompiling ..............................................................................................13
    5.4.3 Dynamic class reloading ........................................................................................14
  5.5 OSGi Web Container ........................................................................................................14
    5.5.1 Java SE considerations ............................................................................................14
    5.5.2 Java EE considerations ............................................................................................14
    5.5.3 Resource lookup .....................................................................................................14
    5.5.4 Resource injection and annotations .........................................................................15
    5.5.5 Web application class loader ..................................................................................15
    5.5.6 Web Container Service ..........................................................................................15
  5.6 Component model interoperability ....................................................................................16
  5.7 Compatibility with the OSGi Http Service .........................................................................16
  5.8 Web Container use of OSGi Services ..............................................................................17
    5.8.1 Web Container use of UserAdmin ..........................................................................17
    5.8.2 Web Container use of LogService ..........................................................................17
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>
<tbody>
<tr>
<td>Initial draft of RFC</td>
<td>Jun 13 2003</td>
<td>David Klein, IBM, <a href="mailto:kleind@us.ibm.com">kleind@us.ibm.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BJ Hargrave, IBM, <a href="mailto:hargrave@us.ibm.com">hargrave@us.ibm.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Thomas Watson, IBM, <a href="mailto:tjwatson@us.ibm.com">tjwatson@us.ibm.com</a></td>
</tr>
<tr>
<td>2nd draft</td>
<td>Sep 17, 2003</td>
<td>David Klein, IBM, <a href="mailto:kleind@us.ibm.com">kleind@us.ibm.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BJ Hargrave, IBM, <a href="mailto:hargrave@us.ibm.com">hargrave@us.ibm.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added section 5.7 to address web descriptor support.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Notes regarding possible changes to this RFC for possible new</td>
</tr>
<tr>
<td></td>
<td></td>
<td>framework capabilities.</td>
</tr>
<tr>
<td>3rd draft</td>
<td>10 Jun. 04</td>
<td>Thomas Watson, IBM, <a href="mailto:tjwatson@us.ibm.com">tjwatson@us.ibm.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed common JSP runtime.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated to take advantage of RFC 70 and 71.</td>
</tr>
</tbody>
</table>
4th draft  11 Oct. 04  Roy Paterson, IBM, rpatersn@us.ibm.com
Finished removing common JSP runtime.
Grammatical fixes.

Draft 5  29 November 2004  Updated to use Declarative Service to register the WebApplication
service rather than a BundleActivator.
BJ Hargrave, IBM, hargrave@us.ibm.com

Draft 6  9 November 2006  Minor updates to reflect R4 has shipped and to use Map instead of
Dictionary.
BJ Hargrave, IBM, hargrave@us.ibm.com

Draft 7  11 November 2008  Rewrote draft after Java EE subcommittee discussions
Subbarao Meduri, IBM, mkv@us.ibm.com
BJ Hargrave, IBM, hargrave@us.ibm.com
Graham Charters, IBM, charters@uk.ibm.com
Thomas Watson, IBM, tjwatson@us.ibm.com

Draft 8  12th January 2009  Subbarao Meduri, IBM, mkv@us.ibm.com
Graham Charters, IBM, charters@uk.ibm.com
BJ Hargrave, IBM, hargrave@us.ibm.com
Adrian Colyer, SpringSource, Adrian.colyer@springsource.com
Rob Harrop, SpringSource, rob.harrop@springsource.com

Draft 9  24th February 2009  URL handler for installing WAR as bundles. Updated draft with feedback
from Jan EEG f2f.

Draft 10  5th March 2009  March 2nd Review feedback.
Clarification on extender interactions via use of @Resource for service
reference
Bugs: 1172, 1181

1 Introduction
The OSGi Web Container provides support for a Web Application written to Servlet 2.5 or later and JSP 2.1 or later. The currently defined OSGi Http Service in the Compendium specification does not meet the needs of Servlet specifications beyond 2.1, and does not fully support the concept of Web applications.

The OSGi Web Container supersedes the functionality of the OSGi Http Service, and can be implemented in a manner that supports bundles written to the original Http Service coexisting with bundles written to the new OSGi Web Container.

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

## 2 Application Domain

Web applications are a critical part of applications built using Java Enterprise Edition (Java EE). The Servlet/JSP model is a popular Web application development model that provides a well known API for developing applications that generate a markup language (e.g., HTML) for rendering by a suitable agent (e.g., web browser). The Java Servlet specification describes how standard web applications function in enterprise environment.

### 2.1.1 Structure of a Web application

A web application is a collection of resources that is typically made up of a collection of some of the following:

- Servlets
- JSPs
- Utility classes
- Static documents (HTML, images, sounds etc.)
- Resource files used by Java classes
- Descriptive meta information that ties the above elements together in web.xml

The following table describes the structure of a Web Application:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Visible web resources such as html, jsp, and images.</td>
</tr>
<tr>
<td>/META-INF</td>
<td>Contains the MANIFEST.MF manifest file, as required for jar files. An application WAR can list its dependencies on external libraries needed in the manifest as described in JAR specification. During deployment of the Web application, the Web container must make the correct versions of the extensions available to the application following the rules defined by the <em>Optional Package Versioning</em> mechanism (<a href="http://java.sun.com/j2se/1.4/docs/guide/extensions/">http://java.sun.com/j2se/1.4/docs/guide/extensions/</a>)</td>
</tr>
<tr>
<td>/WEB-INF</td>
<td>Web application resources that are not visible to the outside world</td>
</tr>
</tbody>
</table>
(i.e. can't be directly accessed by a browser's http request). This directory contains web.xml, the web application deployment descriptor. Note that it is not necessary for every web application to include a web.xml file, especially if the application only contains JSP resources and/or static resources.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/WEB-INF/classes</td>
<td>Java class files of the web app. Classes in this directory must be available to application classloader.</td>
</tr>
<tr>
<td>/WEB-INF/lib/</td>
<td>Library Jar files, which are automatically available to the web app’s classloader. Classes in /WEB-INF/classes take precedence over classes found in WEB-INF/lib jars. Web containers must also be able to recognize declared dependencies expressed in the manifest entry of any of the library JARs under the WEB-INF/lib entry in a WAR.</td>
</tr>
</tbody>
</table>

Web applications can be packaged and signed into a Web Archive format (WAR) file using the standard Java archive tools. Servlet 2.5 specification provides a detailed description of the format of a WAR archive.

### 2.1.2 Web Application deployment descriptor (web.xml)

The Web application deployment includes the following types of configuration and deployment information:

- Initialization parameters for servlets
- Session configuration
- Complex mapping between URIs and servlets
- Application lifecycle listeners
- Servlet filters
- MIME type mappings
- Custom "welcome" pages
- Custom "error" pages
- Customized locale and encoding mappings
- Security role mapping

### 2.1.3 Web application context root

Every Web application running in a server is typically rooted at a unique base URL, also known as the context root (or context path) of the application. The server (web container) uses this initial portion of the URL to determine which web application services an incoming request.

A deployer usually specifies the context root of an application WAR at the time of deployment and allows for customizing the URL space serviced by an application. While an enterprise application has a standard way to specify application context root in a EAR (Enterprise Application Archive), currently there is no provision to specify context root of a web application inside a WAR file. The mechanism of how a context root for a WAR is supported has been vendor-specific.

### 2.1.4 Temporary work area

The Servlet specification requires a temporary storage directory per servlet context (application). The servlet engine must ensure that the content of temporary directory of one servlet context is not visible to servlets contexts of other applications. Given this storage requirement, this RFC mandates that OSGi implementations of this specification MUST have file system support.
2.1.5 JavaServer pages

JSP technology provides a convenient script-like and HTML friendly mechanism that greatly simplifies dynamic content creation. JSP technology is built on top of Servlet technology, and often servlets and JSPs are deployed together in a WAR file. JSPs are compiled into Servlet classes at runtime. The Servlet and JSP specifications describe how JSPs are translated, and managed in a JSP container, and how requests are mapped to JSPs translated into servlets.

3 Problem Description

The OSGi compendium currently includes an Http Service specification. However the design of that specification was developed based upon the Servlet 2.1 specification which predated the concept of the web application in the Servlet 2.2 specification, and as such does not fully support the concept of Web applications. As specified in section 2.1.2, a typical web application can declare various application attributes declaratively in the web.xml file. While in theory, each of these features of web.xml could be implemented programmatically, enterprise developers are used to writing web applications, and there are many tools available for writing them. It would be logical that web applications should be embraced and fully and supported in the OSGi environment.

In practice, the Http Service specification is not really useful for deploying more than a couple of servlets in an application. The Http Service specification also does not provide support for JSPs. We need a specification which supports the current generation of the servlet and JSP specifications and embraces the concept of web applications.

4 Requirements

This specification addresses the following requirements from RFPs 85 & 98:

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

2. The standard MUST NOT require that a particular configuration API or system, including the OSGi Configuration Administration Service, be supported.
3. The solution MUST support deploying web applications as WAR files, as described in the Java Servlet Specification.

4. The solution MUST allow web applications to specify which packages to import from the OSGi Framework.

5. The solution MUST allow web applications to export packages and services to the OSGi Framework, as well as allow OSGi-aware web applications to access other OSGi services.

6. The solution MUST support web applications that support Java Server Pages (JSPs).

7. The solution MUST allow for dynamic modifications to the web application resources, such as JSPs or images without disturbing other parts of the running web application.

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

9. It SHOULD be possible for a Web application bundle to remain installed when its Web Container is dynamically replaced.

10. An OSGi-compliant Web Container MUST NOT be impeded from also being compliant with the Servlet and JSP specifications.

11. An OSGi-compliant Web Container MUST support Servlets and JSPs implemented to the Servlet 2.5 and JSP 2.1 specifications or any later versions of those specifications which are backwards compatible with Servlet 2.5 and JSP 2.1.

12. The OSGi Web Container design MUST NOT require an OSGi Execution Environment greater than that which satisfies the signatures of the Servlet and JSP specifications.

13. An OSGi Web Container MAY provide additional aspects of the technology that are required for Servlet and JSP support to be properly integrated in an OSGi framework but MUST NOT make any syntactic changes to the Java interfaces defined by the Servlet and JSP specifications.

5 Technical Solution

5.1 Architectural overview

Bundles are the deployment and management entities under OSGi. The RFC takes a design approach where a web application is deployed as an OSGi bundle in the framework. There is exactly one web application bundle that corresponds to each deployed web application in the framework.
The specification describes the design requirements for an OSGi Web Container that supports application components written to Servlet and JSP specifications. The Web Container itself is deployed as one or more OSGi bundles. A new service org.osgi.services.webcontainer.WebContainer is introduced for helping with deployment and management of Web applications, and is implemented by the Web container.

The design uses OSGi extender pattern [6], where Web container acts as an extender that is responsible for observing the life cycle of web application bundles. When a web application bundle is started, Web container extender processes the configuration files of the application and instantiates and manages lifecycle of Servlet and JSP components packaged in the web application. The web application bundles thus become managed bundles of the Web container extender.

5.2 Web application life cycle

5.2.1 Installing a Web application

The design supports installing Web applications packaged as:

- Standard WAR file developed per Servlet 2.5 and JSP 2.1 specifications. A standard WAR file needs to be transformed into a valid OSGi bundle before it can be installed into the framework. This may require augmenting the WAR file MANIFEST with OSGi headers such as Bundle-SymbolicName, Bundle-ClassPath etc.
- Standard WAR file that is also built as an OSGi bundle. This eliminates the need for converting a WAR file into a bundle. A deployer may leverage the transformation step to encode web application properties as manifest headers.

Bundles can be installed into OSGi framework using BundleContext.installBundle() API which accepts bundle location argument, typically in the form of a URL. This design proposes using a URL handler for dynamically transforming WAR file into bundle during install. This approach has the following advantages:

- It avoids preprocessing a WAR file as a necessary step before installing.
- It allows for reusing installBundle API for installing WAR files.

The design does not prescribe a specific URL scheme. An implementation is free to define its own URL scheme and register a corresponding URL handler that implements the scheme to install WAR files as bundles. Given that the URL scheme can be private to an implementation, the implementation should provide a utility API to help with the construction of URLs that a deployer can use with installBundle API. The Web container implements a new compendium service org.osgi.services.webcontainer.WebContainer that provides a createWarURL utility method to allow deployer to override various OSGi headers and application properties.

String createWarURL(String location, Map deployOptions, Map deployDefaults);

- location - represents WAR URL
- deployOptions - a Map representing name/value pairs of attributes that the deployer wants to override in the bundle.
- deployDefaults - a Map representing name/value pairs of attributes that deployer wants to use in the bundle only if those attributes are not specified in the bundle. This allows for honoring values in the bundle, and use deployer specific values only in cases where a default value is not specified in the bundle. (NOTE: This option is still under discussion and subject to change.)
- Returns - String representation of the URL that encodes various deployer options.

Each attribute represents an OSGi manifest header or an application property. Following is the list of valid attributes and their values:
<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Type</th>
<th>OSGi header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constants.Bundle_SymbolicName</td>
<td>String</td>
<td>Bundle-SymbolicName</td>
</tr>
<tr>
<td>Constants.Bundle_Version</td>
<td>String</td>
<td>Bundle-Version</td>
</tr>
<tr>
<td>Constants.Bundle_ManifestVersion</td>
<td>String</td>
<td>Bundle-ManifestVersion</td>
</tr>
<tr>
<td>Constants.Bundle_ClassPath</td>
<td>Array of string objects</td>
<td>Bundle-ClassPath</td>
</tr>
<tr>
<td>Constants.Import_Package</td>
<td>Array of string objects</td>
<td>Import-Package</td>
</tr>
<tr>
<td>Constants.Export_Package</td>
<td>Array of string objects</td>
<td>Export-Package</td>
</tr>
<tr>
<td>WebContainer.Web_ContextPath</td>
<td>String</td>
<td>Web-ContextPath</td>
</tr>
<tr>
<td>WebContainer.Web_JSPExtractLocation</td>
<td>String</td>
<td>Web-JSPExtractLocation</td>
</tr>
</tbody>
</table>

Implementations may ignore any attributes that they do not recognize.

An implementation may also publish its URL scheme so that a deployer can directly code a URL and use with the `installBundle` API. As an example, The PAX Web Extender [7], uses a “war:” URL scheme to install WAR files as OSGi bundles in the framework.

### 5.2.1.1 WAR manifest processing

An implementation must register a URL handler for processing implementation specific WAR URLs. When a deployer invokes `installBundle` to install WARs using the URL generated by the `createWarURL` method, the URL handler gets control, and examines the input URL and adds several manifest entries on the fly based on the deployer specified options. The URL handler uses the following while computing manifest header values:

- WAR file manifest entries – developer defaults
- Properties supplied by deployer via install URL (e.g., query string) - deployer defaults
- Compute based on additional criteria, e.g., other artifacts in the archive.

The WAR manifest headers are modified as described below:

1. **Bundle-Version: optional**
   - If `Constants.Bundle_Version` deployer option is specified in the URL, it will be used.
   - Otherwise, if a `Bundle-Version` header exists in the manifest, it will be preserved.
   - Otherwise, if `Constants.Bundle_Version` deployer default option is specified in the URL, it will be used.

2. **Bundle-ManifestVersion: Required**
   - Must be >= 2
   - If `Constants.Bundle_ManifestVersion` deployer option is specified in the URL, it will be used.
   - Otherwise, if a `Bundle-ManifestVersion` header exists in the manifest, it will be preserved.
   - Otherwise, if `Constants.Bundle_ManifestVersion` deployer default option is specified in the URL, it will be used.
   - Otherwise, set it to 2 (default)

3. **Bundle-SymbolicName: Required**
   - If `Constants.Bundle_SymbolicName` deployer option is specified in the URL, it will be used.
Otherwise, if Bundle-SymbolicName header already exists in the manifest, it will be preserved. Otherwise, if Constants.Bundle_SymbolicName deployer default option is specified in the URL, it will be used. Otherwise, a unique symbolic name is derived based on artifacts in a implementation specific way.

4. Bundle-Classpath: Required

Add or update Bundle-ClassPath header by

a. initializing the first path entry to “WEB-INF/classes/” if it is not already present on path.
 b. Adding each of the libraries from “WEB-INF/lib” if not already present on path. Order is unspecified.

If Constants.Bundle_ClassPath deployer option is specified in the URL, it will be appended to the path.

5. Import-Package: Required

Add or update Import-Package header by adding these imports if not already present.

javax.servlet; version=2.5
javax.servlet.http; version=2.5
javax.servlet.jsp; version=2.1
javax.servlet.jsp.tagext; version=2.1

If Constants.Import_Package deployer option is specified in the URL, add imports specified by the deployer to Import-Package header.

An implementation may also analyze classes in the application archive to compute the imports needed by classes and add them to Import-Package header.

6. Export-Package: Optional

If Constants.Export_Package deployer option is specified in the URL, add those exports imports to Export-Package header.

7. Web-ContextPath: Optional

Specifies the context path of the web application.

If WebContainer.Web_ContextPath deployer option is specified, it will be used.
Otherwise, if the header exists in the manifest, it will be preserved.
Otherwise, if WebContainer.Web_ContextPath deployer default option is specified in the URL, it will be used.
Otherwise, a unique context path value is derived in implementation specific ways.

8. Web-JSPExtractLocation: Optional

Specifies the location where JSP files are extracted before translation. If the container supports JSP reloading, a deployer can directly modify the JSP files from this location, and expect the container to retranslate, recompile and reload those classes at runtime whenever they are touched.

If WebContainer.Web_JSPExtractLocation deployer option is specified, it will be used.
Otherwise, none. A container may derive such a location uniquely per application.

Once a WAR archive is installed as OSGi bundle, its life cycle is managed just like any other bundle in the framework.

5.2.2 Starting a web application

A web application is started by starting its corresponding web application bundle. The Web container extender listens for bundle life cycle events. A web application and the web container extender may start in any order. The web container bundle recognizes a Web application bundle by looking for the presence of one or more of the following:

- Web-contextPath manifest header
- Web.xml deployment descriptor
- Bundle’s location has a .war extension

Or any other implementation specific criteria. Note that it is not mandatory for all WAR files to contain a web.xml, especially if the application only contains static resources, or JSP files.
The Web container processes deployment information by processing web.xml descriptor and any deployment specific annotations specified in the application. The descriptor may specify a metadata-complete attribute, whose value, when set to “true”, indicates that the class files and Jar files of the application need not be examined for deployment specific annotations. Annotation scanning itself should not result in loading of the classes, such that it prevents further byte code weaving of the classes.

As web applications are modularized further into multiple bundles (and not deployed as WAR files only), it is possible that a web application bundle can have (import) dependencies on other deployed bundles. The container will be required to scan those dependant bundles for annotations in such cases.

The container performs the necessary initialization of web components in the bundles, as described in Servlet 2.5 specification. This involves the following:

- Create ServletContext for the application.
- Instantiate configured servlet event listeners.
- Instantiate configured application filter instances etc.

The Web container is required to complete instantiation of listeners prior to start of execution of the first request into the application. Attribute changes to ServletContext and HttpSession objects may occur concurrently. The 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.

5.2.3 Stopping web application

A web application is stopped by simply stopping the corresponding web application bundle. In preparation for shutdown, the web container notifies configured listeners in the reverse order to their declaration in web.xml. Per servlet 2.5 specification, session listeners are notified first, followed by context listeners of the application shut down.

5.2.4 Uninstalling web application

A web application can be uninstalled by simply stopping the corresponding web application bundle. The application will be permanently removed from the framework.

5.3 Accessing OSGi environment from Web application

In order to properly integrate in an OSGi environment, a Web application may need access to OSGi service registry for publishing its services and accessing services provided by other bundles. This may require an OSGi-aware web application access to BundleContext object.

To facilitate this, Web container will make BundleContext instance available to the web application via the ServletContext interface using a special osgi-bundlecontext attribute. Here is an example of how a Servlet can obtain BundleContext:

```java
BundleContext ctxt = (BundleContext)ServletContext.getAttribute("osgi-bundlecontext");
```

Alternatively, a web component can obtain BundleContext provided it has access to the Bundle object (e.g., Bundle.getBundleContext()). A general mechanism for obtaining the Bundle object from any class loaded by that bundle is being proposed through Bug 786.
5.4 JavaServer Pages support

JavaServer Pages is a popular rendering technology that simplify page construction. This RFC supports JSP 2.1 specification. A web component is either a servlet or a JSP page. The servlet element in a web.xml deployment descriptor is used to describe both types of web components. JSP page components are defined implicitly in the deployment descriptor through the use of an implicit "jsp" extension mapping, or explicitly through the use of a jsp-group element. Supporting JavaServer pages in a web application bundle poses some unique challenges as discussed below:

5.4.1 In-line compiling

A unique aspect of JSPs is that the web container compiles a JSP page into a servlet class either during application deployment phase, or at the time of request processing, and dispatches the request to a servlet object of such dynamically created class. Often times, the compilation task is delegated to a separate JSP compiler component, who will be responsible for identifying the necessary tag libraries, and generating the corresponding servlet class file. The container then proceeds to load the dynamically generated class, creates a servlet instance and dispatches the request to the servlet.

Supporting in-line compilation of a JSP page inside a bundle will require that the container should maintain a private area where it can store such compiled classes. The framework provides a private persistent storage area for each installed bundle, and an implementation may leverage (but is not required to) the private storage area for this purpose. The container may keep the generated classes around persistently for the life of the application. The container may be required to construct a special classloader to load generated JSP classes such that classes from the bundle class path are visible to newly compiled JSP classes.

5.4.2 Dynamic recompiling

In Java EE web application world, it is a common expectation that JSPs of a web application can be modified on the fly once an application is deployed, and that the container will recognize JSPs that are updated, recompile and reload them on subsequent requests, without the need for web application to be redeployed.

While it is less common for a web application in a production environment to dynamically update JSP pages, scenarios exist where it is very useful to dynamically compile and load JSP pages without uninstalling and redeploying the application. For example:

- During application development, it would be useful to be able to modify JSP pages, and test the changes without redeploying the entire application.
- Certain Portal type applications have the ability to dynamically generate JSP pages as part of their execution, and use the generated JSPs to render new composite pages.

Dynamic reloading support will require that the deployer has access to the bundle artifacts post deployment so that JSP artifacts can be modified after the application is deployed. Most popular web containers support this model in implementation specific ways. The following pattern is used in most cases.

- The deployer optionally can specify a file system location where JSP files will be extracted before compiling. The location is typically unique for each web application.
- The deployer may update the JSP file in the location after the application has started. Upon the next service request on the JSP, the container detects that JSP file is refreshed (by comparing the time stamp), and recompiles it by invoking the JSP compiler, this time passing the URL to the JSP file in the file system location instead of the archive file.

A deployer can control the extract location of JSPs by specifying WebContainer.Web JSPExtractLocation attribute. Development tooling is the main use case for JSP dynamic recompiling, and by standardizing on the location, development tools can consistently work with multiple web container implementations.
5.4.3 Dynamic class reloading

The container will need to reload a servlet class that is generated by compiling a JSP. Further, it is necessary to ensure that the compiled JSP class has visibility to classes in the application bundle’s class path. Several techniques can be used by a container implementation to achieve this.

- If a container supports dynamic compiling only, but not reloading, it is possible to pre-construct a Bundle-Classpath header such that newly created classes will be visible through the bundle classloader.

- It is possible that the container may construct a specialized classloader whose parent is set to the bundle classloader, and load JSP compiled classes using the special classloader.

- To support dynamic reloading of individual JSPs, the container may choose load each of the JSPs via a specialized class loader as described above to load generated servlet classes, and throw away the classloader whenever JSP needed to be reloaded on detecting modifications.

5.5 OSGi Web Container

The specification defines an OSGi Web container implementation as one or more OSGi bundles that collectively implement Servlet 2.5 and JSP 2.1 specifications. The following section describes requirements for an OSGi web container.

5.5.1 Java SE considerations

The Servlet 2.5 specification requires J2SE 5.0 as minimum execution environment. Consequently, it would be the minimum execution environment for running the OSGi Web container.

5.5.2 Java EE considerations

A OSGi web container implementation will need to consider additional requirements in order to be Java EE compliant. Servlet 2.5, JSP 2.1 together with Java EE 5 specification describes a comprehensive set of requirements that a Java EE compliant web container must implement.

In practice, a Web container implementation supports a subset of Java EE services and makes them available to applications, even when it is not fully compliant with Java EE specification. These include support for

- Application name spaces
- Environment naming context (e.g., java:comp/env access)
- Transactions

This specification highly recommends (but does not require) that an OSGi web container provide integration with a

- Integration with a Transaction manager if available (described via RFC 98)
- Integration with JNDI (described in RFC 142)
- Integration with Persistence provider (RFC 115)

5.5.3 Resource lookup

The ServletContext interface provides direct access only to the hierarchy of static content documents that are part of the Web application, including HTML, GIF, and JPEG files, via the getResource() and getResourceAsStream() methods of the ServletContext interface. These methods are not used to obtain dynamic content. For example, in a container supporting the JavaServer Pages specification, a method call of the form getResource("/index.jsp") would return the JSP source code and not the processed output. The full listing of the resources in the Web application can be accessed using the getResourcePaths(String path) method.
These resources will not be on the Bundle classpath, and the Web container can invoke Bundle.getEntry() and Bundle.findEntries() methods correspondingly to service such requests on ServletContext interface.

5.5.4 Resource injection and annotations

The web application deployment descriptor may specify an attribute `metadata-complete` attribute on the web-app element. This attribute defines whether the web descriptor is complete, or whether the class files of the jar file should be examined for annotations that specify deployment information. If `metadata-complete` is set to `true`, the container must ignore any Servlet annotations present in the class files of the application. If the full attribute is not specified or is set to `false`, the deployment tool should examine the class files of the application for annotations.

An OSGi web container that is also compliant with Java EE 5 should support the following annotations.

```
@Resource
@Resource
@DeclareRoles
@PreDestroy
@PostConstruct
@PersistenceContext
@PersistenceContexts
@PersistenceUnits
```

If supported, annotations must be processed in application classes that implement the following interfaces.

- Javax.servlet.Servlet
- Javax.servlet.Filter
- javax.servlet.ServletContextListener
- javax.servlet.ServletContextAttributeListener
- javax.servlet.ServletRequestListener
- javax.servlet.ServletRequestAttributeListener
- javax.servlet.http.HttpSessionListener
- javax.servlet.http.HttpSessionAttributeListener

5.5.5 Web 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.

5.5.6 Web Container Service

The OSGi web container implements a Web container service, and register it in the service registry. This service provides APIs for installing WAR files, and interrogate the Web Container regarding the state of web applications currently deployed in the framework.

```
public interface org.osgi.service.webcontainer.WebApplication {
    String getContextPath();
    ServletContext getServletContext();
    Bundle getBundle();
}
```

```
public interface org.osgi.service.webcontainer.WebContainer {
    Public static final String Web_ContextPath="Web-ContextPath";
    Public static final String Web_JSPExtractLocation="Web-JSPExtractLocation";

    String createWarURL(String location, Map deployOptions, Map deployDefaults);
```
5.6 Component model interoperability

It is conceivable that a web application may need to interoperate with components written to other component models, such as a Declarative Services specification or Blueprint services (RFC 124). Per Servlet spec, the web container owns the life cycle of the servlet. This eliminates the possibility of a servlet component to simultaneously be a component of another component model.

A typical interaction pattern is that a servlet may depend on external service provided by a BluePrint service component. It is possible that the servlet may not be put into service until the dependant service is available, and may require injection of a service into the servlet component. This may require coordination between extenders of different component models. The OSGi component model is dynamic, and it is simply not possible to order the execution of extenders (there is no ordering in OSGi). Modeling extender interactions as service dependencies would be a clean way to facilitate extenders interactions. For servlets, the problem will be reduced to how a OSGi service can be injected into a servlet component.

The servlet specification requires support JSR 250 annotations for Java EE compliant web containers, and of particular interest is the support for injecting resources. The "@Resource" annotation allows containers to inject JNDI resources into the components. This means, so long as OSGi services are accessible as JNDI resources, services can be injected into servlet components using the resource injection mechanism as described in the servlet specification.

The JNDI Integration (RFC 142) describes how OSGi services can be made available via JNDI context. It defines a "osgi:services" name space and leverages URL context factory pattern to facilitate JNDI integration with OSGi service registry.

To summarize, the recommended interaction pattern between web components with other component models is:

- Model component model interactions as dependencies at the OSGi service registry layer
- Make OSGi services available via JNDI context
- Use Java EE resource injection mechanism to inject OSGi services into web components

The downside of this approach is that this requires that a web container must support annotations, which is not a strict requirement unless the container needs to be Java EE complaint.

NOTE: The following sections are still under discussion, no agreements have been made yet.

5.7 Compatibility with the OSGi Http Service

An OSGi Http Service (see OSGi R4 compendium specification) may be implemented using the Web container implementation to provide compatibility with OSGi Http Service. If this implementation approach is used then an OSGi Http Service and the OSGi Web Container can co-exist on the same platform and access the same ports. Note that this implementation approach is a RECOMMENDED approach and is not required to comply with the OSGi Web Container specification.
5.8 Web Container use of OSGi Services

The Web Container may use standard OSGi services to implement some of the features of the Servlet 2.5 specification. The following OSGi services are RECOMMENDED for implementing parts of the Servlet 2.5 specification.

- UserAdmin Service - Used to authenticate Web Application users.
- LogService - To write Servlet logs.
- SAXParserFactory - To process web descriptors.

5.8.1 Web Container use of UserAdmin

Web Application resources that have security constraints must only be accessed by authenticated users (see Servlet 2.3 specification). To authenticate users the UserAdmin service may be used. The password credentials for each user should be stored with the key "password". For example, to get the password for a user from UserAdmin the following code can be used:

```java
Role role = userAdmin.getRole(username);
String password = null;
if (role instanceof User) {
    User user = (User) role;
    // get the password, Here we assume the password is stored a the
    // credential "password".
    password = (String)user.getCredentials().get("password");
}
```

When security constraints are set for a Web Application resource a role-name can be set. For example, the following web descriptor tags sets a security constraint for a whole Web Application that allows any valid OSGi user to have access:

```xml
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Web Log Example</web-resource-name>
        <description>
            An example of a Web Application with a security-constraint
        </description>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <description>Any valid OSGi User</description>
        <role-name>user.anyone</role-name>
    </auth-constraint>
</security-constraint>
```

The role-name can be an individual UserAdmin User or Role. In the above example user.anyone is the default Role that all UserAdmin Users have (see OSGi UserAdmin specification).

5.8.2 Web Container use of LogService

The OSGi LogService may be used to store any logs that servlets write. A Servlet can log messages by calling the GenericServlet.log() or the ServletContext.log() methods (see Servlet 2.3 specification). Calls to of the log() methods may be logged to an OSGi LogService.

5.8.3 Web Container use of SAXParserFactory

The web descriptor file /WEB-INF/web.xml is an XML file (see Servlet 2.3 specification). A SAX parser may be obtained by using a SAXParserFactory service to process the XML file.
6 Requirements not considered

Following requirements are not considered for the design of this specification.

- The solution SHOULD also support deploying web applications in exploded form from a file system directory, following the format of the WAR file from the Java Servlet Specification.

- The solution MUST support deploying in-memory web applications. For this type of deployment, the bundle deploying a web application will be expected to supply an XML document conforming to the “web.xml” format defined in the Java Servlet Specification, plus a mechanism for loading classes and static resources for the web application. For instance, the interface could require a DOM tree and a ClassLoader object, or some other set of standard programming artifacts.

7 Security Considerations

7.1 Web Container bundle

The Web Container should only be implemented by a trusted bundle. This bundle requires the following security permissions.

- ServicePermission[get] for the org.osgi.service.log.LogService interface. This allows the Web Container to log servlet messages.

- ServicePermission[get] for the org.osgi.service.useradmin.UserAdmin interface. The UserAdmin service can be used to implement the J2EE declarative security model.

- ServicePermission[get] for the javax.xml.parsers.SAXParserFactory interface. The SAXParserFactory service can be used to process Web Application Web descriptors.

- UserAdminPermission[getCredential] for the credential "password". Used to lookup user passwords for the J2EE declarative security model.

- AdminPermission[resource,class] to call Bundle.getEntry, Bundle.getEntryPaths and Bundle.loadClass on the Web Application Bundles.
8 Document Support

8.1 References

[2]. Servlet 2.5 specification
[3]. JSP 2.1 specification
[4]. Java EE 5 specification
[5]. RFC 124 (Blueprint services)
[7]. PAX Web Extender http://wiki.ops4j.org/display/ops4j/Pax+Web+Extender

8.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Subbarao Meduri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>4205 Miami Blvd, Durham, NC USA</td>
</tr>
<tr>
<td>Voice</td>
<td>+1 919 254 5210</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:mkv@us.ibm.com">mkv@us.ibm.com</a></td>
</tr>
</tbody>
</table>

8.3 Acronyms and Abbreviations

WAR – Web Application Archive
JSP – Java Server Page

8.4 End of Document
RFC 98 Transactions in OSGi

Draft

16 Pages

Abstract
An increasing number of service specifications in the OSGi Service Platform rely on some form of transactional behaviour. Other service specifications could improve if they had transactional behaviour. This RFC defines a transaction model and identifies Java transaction APIs for use in OSGi environments, including embedded and constrained environments.
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 ..................................................................... 4
  2.1 Recovery ............................................................................ 6
  2.2 Java Transaction Architecture ........................................... 6
  2.3 Why Transactions .............................................................. 7

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

Requirements .................................................................................. 8

4 Technical Solution ....................................................................... 9
  4.1 Using JTA ................................................................. 9
  4.2 Compliance ................................................................. 9
  4.3 Components of the Transaction Service ............................... 9
    4.3.1 Transaction Originator ............................................ 10
    4.3.2 Transaction Manager ............................................. 10
    4.3.3 Volatile Resources ................................................ 10
    4.3.4 Transaction Resources .......................................... 11
  4.4 Locating OSGi transaction services .................................... 11
  4.5 Use Cases .......................................................................... 11
    4.5.1 Create Transaction ................................................ 11
    4.5.2 Join Transaction ..................................................... 11
    4.5.3 Commit Transaction ............................................... 11
    4.5.4 Prepare Resource .................................................. 12
    4.5.5 Commit Resource .................................................. 12
    4.5.6 Rollback Transaction .............................................. 12
    4.5.7 Rollback Resource ................................................ 12
    4.5.8 Assign Transaction to Thread .................................. 12
  4.6 Functionality ....................................................................... 12
    4.6.1 Scope of a global transaction .................................. 12
    4.6.2 Correctness of the State .......................................... 12
    4.6.3 End of Transaction ................................................ 13
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>
<tbody>
<tr>
<td>Initial</td>
<td>Aug 30 2004</td>
<td>Peter Kriens</td>
</tr>
<tr>
<td>rewriting</td>
<td>Oct 30 2007</td>
<td>Pavlin Dobrev</td>
</tr>
<tr>
<td>0.2</td>
<td>Oct 31 2007</td>
<td>Apply comments from Valentin Valchev</td>
</tr>
<tr>
<td>0.3</td>
<td>Nov 08 2007</td>
<td>Pavlin Dobrev – minor misspellings</td>
</tr>
<tr>
<td>0.4</td>
<td>Nov 22 2007</td>
<td>Peter Kriens – edit and added comments</td>
</tr>
<tr>
<td>0.5</td>
<td>Nov 29 2007</td>
<td>Pavlin Dobrev – answer on some comments</td>
</tr>
<tr>
<td>0.6</td>
<td>Dec 13 2007</td>
<td>Pavlin Dobrev – answer to Thomas Watson <a href="mailto:tjwatson@us.ibm.com">tjwatson@us.ibm.com</a> comments</td>
</tr>
<tr>
<td>0.7</td>
<td>Jul 09 2008</td>
<td>Ian Robinson/Roman Roelofsen – rebase on JTA.</td>
</tr>
<tr>
<td>0.71</td>
<td>6 Aug 2008</td>
<td>Prepare for public draft.</td>
</tr>
<tr>
<td>0.8</td>
<td>21 Nov 2008</td>
<td>Address bugs: 891, 892, 893, 894, 895, 907, 908, 920, 924</td>
</tr>
<tr>
<td>0.9</td>
<td>12 Dec 2008</td>
<td>Address bugs: 950, 948, 959, 952, 949, 946, 953, 954, 947, 951,</td>
</tr>
</tbody>
</table>
1 Introduction

An increasing number of APIs in the OSGi are requiring transactional concepts. This is to be expected because transactions can simplify applications that have to run in a dynamic and distributed world. The OSGi expert groups had earlier discussions regarding transactions but at that time (1999) transactions were deemed too heavy and complex to add to the specifications. This RFC suffered the same fate in 2004 for the OSGi Release 4 due to lack of time. However, OSGi R5 seems to be the appropriate time to re-discuss this because of strong requirement for transactions in EEG.

This RFC introduces the transaction concepts and outlines the different trade-offs that need to be made in the API.

2 Application Domain

For the purposes of this specification, a transaction is a coordinated series of changes to one or more information stores. In almost any reasonable case, multiple closely related changes are required for a transaction; these changes can depend on external or internal values. This quickly introduces the problem of how to keep the system in a consistent state when there are unexpected failures and multiple parties that may change the same information. Transaction processing systems that address these problems have been at the heart of business computing systems since the early sixties. There are many different types of business transaction but this specification is concerned with those that have the following ACID properties (adapted from [2]):

1. Atomic – The transaction completes successfully ( commits ) or if it fails ( aborts ) all of its effects are undone (rolled back). The changes include database changes, messages and actions on actuators.
2. Consistency – A transaction is a correct transformation of the state. The actions taken as a group do not violate any of the integrity constraints associated with the state. This requires that a transaction is a correct program.
3. Isolation – Intermediate states produced while a transaction is executing are not visible to other transactions. Furthermore transactions appear to execute serially, even if they are actually executed concurrently.
4. Durability – Once a transaction completes successfully ( commits ), its changes to the state survive failure.

Trying to achieve these properties in a program without proper assistance of the environment is difficult. It therefore became clear quickly to the pioneers in this area that support was needed. At first this was embedded in the database because the problems are most visible in the persistent storage. However, this turned out to be
insufficient when multiple persistent stores were involved in the same transaction. For example, a money-transfer operation between two accounts is a transactional operation requiring both the credit and debit parts of the transfer to succeed.

A “local transaction” is one that involves a single information store (resource manager). A “global” or “distributed transaction” is one that may involve two or more resource managers. (The term “distributed” here does not necessarily imply that the transaction spans multiple execution processes – it is “distributed” from the perspective of the resource manager. Both terms are used interchangeably within the literature. This specification will refer to “global” transactions rather than “distributed” transactions).

A global transaction requires a transaction manager that is logically external from the resource managers to coordinate their joint outcome. The common model that evolved over the years is the “two phase commit” (2PC) model with resource managers being directed by the transaction manager. A concrete 2PC protocol implemented by all popular commercial resource managers and transactions is the XA protocol [3]. In the 2PC model, a transaction is started and the program will perform the steps to execute the transaction. Operations on transactional data occur in subsystems such as databases which are the transactional resource managers. Once a resource manager is accessed as part of a global transaction, it “joins” the current transaction so that the transaction manager is aware that the resource manager needs to participate in the outcome of the transaction. But what is the current transaction? One possibility for this would be to pass a transaction object in each call. However, this is error prone and cumbersome for the programmer. Since the days of multithreaded server environments, the usual model is that a transaction context is associated with the current thread of execution. All calls that are execution in a thread are then assumed to be part of the transaction. A data operation on a certain thread then implies a data operation in the context of a specific transaction and the resource manager for the data store joins that specific transaction for outcome coordination and manages visibility to resource updates in the context of that transaction. A purely thread-based transaction context is obviously not sufficient for distributed systems where the different systems should be part of the same transaction. Specifications such as the CORBA Object Transaction Service (OTS) [4] and WS-AtomicTransaction [5] defines mechanisms by which transaction contexts are implicitly propagated on remote requests over IIOP and SOAP/HTTP transports respectively, to be used by the transaction manager in the target system to associate the work of the local thread with the overall distributed transaction.

At the end of a successful transaction the application program must decide whether to initiate a commit or rollback request for all the changes made within the scope of the the transaction. The program requests that the transaction manager completes the transaction and the transaction manager then negotiates with the resource managers to reach a coordinated outcome. During the first (prepare) phase, an individual resource manager must make durable any state changes that occurred during the scope of the transaction, such that these changes can either be rolled back or committed later once the transaction outcome has been determined. Assuming no failures or vetoes to the successful outcome occurred during the first phase, in the second (commit) phase resource managers may “overwrite” the original state with the state made durable during the first phase. If just one of the resource managers vetoes during the prepare phase then the transaction rolls back.

In order to guarantee consensus, strict two-phase commit is necessarily a blocking protocol: after returning the first phase response, each participant who returned a commit response must remain blocked until it has received the coordinator’s phase 2 message. Until they receive this message, any resources used by the participant are unavailable for use by other transactions, since to do so may result in non-ACID behavior. If the coordinator fails before delivery of the second phase message these resources remain blocked until it recovers. To break this blocking nature, participants that have got past the prepare phase are allowed to make autonomous decisions as to whether they commit or rollback: such a participant must record this decision in case it is eventually contacted to complete the original transaction. If the coordinator eventually informs the participant of the transaction outcome and it is the same as the choice the participant made, then there’s no problem. However, if it is contrary, then a non-atomic outcome has obviously happened: a heuristic outcome. How this heuristic outcome is reported to the application and resolved is usually the domain of complex, manually driven system administration tools,
since in order to attempt an automatic resolution requires semantic information about the nature of participants involved in the transactions.

2.1 Recovery
An important aspect of transaction atomicity is *recovery processing*. Recovery processing may be required after a transaction manager or resource manager fails unexpectedly during a transaction. This can happen at any moment in time, including between the prepare and the commit phase. The XA 2PC protocol is, by definition, a *presumed-abort* protocol. This means that the transaction manager and resource managers all agree up-front that any failure that occurs before the prepare phase can be assumed to result in rollback. This gives resource managers the right to unilaterally rollback before they are prepared. It also means that a transaction manager does not need to persist any information about the transaction before it has completed the prepare phase.

Recovery processing is required following a failure of the transaction manager to resolve any parts of a global transaction that were prepared but not completed at the point when the failure occurred. Resource managers with prepared work may be holding locks and need to be directed to commit or rollback their work. The XA 2PC protocol defines a recovery protocol between the TM and RMs to resolve such “in-doubt” work.

2.2 Java Transaction Architecture
As mentioned above, the most ubiquitous standard 2PC protocol between a transaction manager and a resource manager is the XA protocol [3]. The Java Transaction Architecture (JTA) [6] defines mappings of the XA protocol to local Java interfaces. It specifies the means for XA-compliant resource managers to be coordinated through a process-local Java “XA resource adapter” by a Java transaction manager.

A JTA transaction manager implements the interfaces of the javax.transaction package which contains the key interfaces for transaction management: TransactionManager, UserTransaction, and Transaction. A transactional resource manager implements the interfaces of the java.x.transaction.xa package that contains the XAResource interface. The model in JTA is that resource managers (instances of XAResource) enlist with the TransactionManager after which they participate in the transaction.

JTA is a Java mapping of the XA interface. It supports all the well-known optimizations of two-phase commit and offers a mature and widely implemented means to support transactions in a Java runtime.
2.3 Why Transactions

The concept of atomic transactions has played a cornerstone role in creating today’s enterprise application environments by providing guaranteed consistent outcome in complex multiparty business operations and a separation of concerns in applications yielding well designed business process implementations. So just what is an atomic transaction (often abbreviated to just transaction)? Put simply, a transaction provides an “all-or-nothing” (atomic) property to work that is conducted within its scope, whilst at the same time ensuring that shared resources are isolated from concurrent users. Importantly application programmers typically only have to start and end a transaction; all of the complex work necessary to provide the transaction’s properties is hidden by the transaction system, leaving the programmer free to concentrate on the more functional aspects of the application at hand.
3 Problem Description

Transaction semantics obviously provide a significant increase in robustness and simplicity. ACID properties can be guaranteed on both successful and failed execution. In current OSGi applications, this robust exception model is absent and it is left to the implementers of the different subsystem. In many cases it is virtually impossible to correctly clean up in the light of failures.

This problem is very visible, for example, in the mobile specifications that are being developed for R4. There is currently a proliferation of “transaction” like APIs in the mobile specifications. This proliferation causes:

- Duplication of effort and code on the platform.
- It is also likely that it creates quality problems because implementing transactions is a complex task without centralized support.
- Requires that the transaction coordination is handled by the application programmer because application subsystems cannot find a current transaction to join.
- Increases the learning curve for application programmers due to the different semantics associated with the different subsystems
- There is no central overview of the transactional state of the system. This seriously hinders diagnosing and debugging systems as well as help desks.

There is therefore a need to centralize the management of transactions. This must minimize the code size, increase reliability, and enable tools.

Requirements

Provide a comprehensive model that allows components in an OSGi Service Platform to perform their actions in a transactional way.

1. Identify the Java APIs that must be provided by a Transaction Manager to delineate a transaction boundary and provide a means for resources to join a transaction. Identify the Java API that must be provided by a resource manager to support two-phase commit, including recovery.

2. The transaction API must be suitable both for enterprise runtimes and embedded systems

3. Reuse existing widely used Java transaction technology wherever possible and avoid repeating what is already specified elsewhere.
4. The specification should place no requirements on a transaction service implementation to be recoverable. It should be noted, however, that a transaction service implementation can only provide transaction atomicity if it supports recovery processing following a failure.

5. The transaction service specification must allow implementation of transactional applications without the need for external changes to the application’s business interfaces and configuration.

4 Technical Solution

4.1 Using JTA

The existing JTA specification addresses the requirements stated above; in addition many providers of Java resource adapters already implement the JTA XAResource interface. The JTA specification [6] defines all the Java transaction interfaces and semantics for transaction and resource managers and this specification reuses those APIs as-is with no modification. The XA+ specification [3] defines the semantics of the underlying XA protocol. This specification defines only additional information related specifically to the OSGi architecture, such as how a transaction service reference can be discovered in the OSGi service registry.

4.2 Compliance

This specification defines an OSGi transaction service implementation as one or more OSGi bundles that collectively implement the classes and interfaces of the javax.transaction package defined in [6]. A compliant OSGi transaction service implementation must pass all the tests defined by the OSGi transaction service compatibility suite. A compliant OSGi transaction service implementation is not required to be certified as a compliant JTA implementation, although it may optionally be certified as compliant to the JTA specification when used as part of a Java EE profile.

4.3 Components of the Transaction Service

There are four basic roles for transaction support in the OSGi framework:

1. **Transaction originator.** This may be either an application role or a framework role and it is responsible for demarcating transactional units of work. Application components use the JTA UserTransaction interface to demarcate transaction contexts.

2. **Transaction manager.** The transaction manager provides the implementation of transaction capability. It responds to requests to demarcate transaction contexts, associates transaction contexts with the local thread of execution, potentially distributes/receives transaction contexts on remote requests, accepts the registration of transactional participants and coordinates participants to an atomic outcome. The transaction manager is the core of the OSGi transaction support and MUST implement the javax.transaction.UserTransaction, javax.transaction.TransactionManager, javax.transaction.Transaction and javax.transaction.SynchronizationRegistry interfaces as defined in [6]. The transaction manager is a logical component consisting of one or more OSGi services which are registered in and available through the OSGi service registry as described in 4.4.
3. **Volatile resources.** Some objects have an interest in the outcome of the transaction but do not participate in 2PC, for example persistent caches that need to be flushed at the end of the transaction immediately prior to 2PC. These objects implement the javax.transaction.Synchronization interface and are enlisted in the transaction via the TransactionSynchronizationRegistry.registerInterposedSynchronization(Synchronization).

4. **Transactional resources.** Resource managers that participate in 2PC provide an implementation of the javax.transaction.xa.XAResource interface. XAResources are enlisted in a transaction via the Transaction.enlistResource(XAResource) interface.

### 4.3.1 Transaction Originator

A transaction is requested to begin and end (commit or rollback) by the transaction originator and, once started, a transaction context is associated with the thread of its originator. Application components begin a global transaction using the begin() method of UserTransaction. Transactions may also be originated by framework components using the javax.transaction.TransactionManager interface. This is a richer interface than the UserTransaction interface and provides additional transaction context management operations such as suspend() and resume() that are not appropriate for application use.

**Limitations:**

- A global transaction may only be associated with a single thread at any point in time. The specific thread to which a transaction is associated may change over time. A thread may have no more than one global transaction concurrently associated with it.

### 4.3.2 Transaction Manager

The transaction manager is a logical component consisting of one or more OSGi services and provides transactional capabilities for the framework. It

- Provides the means for a transaction originator to demarcate transactional boundaries, through the javax.transaction.UserTransaction and javax.transaction.TransactionManager interfaces.

- Provides a means to represent an instance of a specific transaction using a javax.transaction.Transaction object, obtained by calling the getTransaction() method of a TransactionManager object.

- Maintains an association between a Transaction object and a thread of execution such that the Transaction is associated with at most one thread at a time.

- Accepts enlistment of volatile and transactional resources in a specific transaction.

- Notifies volatile resources of the outcome of the transaction.

- Coordinates transactional resources using the two-phase commit protocol at the end of the transaction.

- Drives the recovery interface of transactional resource following a failure of the transaction manager to ensure the atomic completion of transactional work.

### 4.3.3 Volatile Resources

Volatile resources are components that do not participate in 2PC but are called immediately prior to and after 2PC. If a request is made to commit the transaction then the volatile participants have the opportunity to perform some “beforeCompletion” processing such as flushing cached updates to persistent storage. Failures during beforeCompletion will cause the transaction to rollback. In both the commit and rollback cases the volatile
resources are called after 2PC to perform “afterCompletion” processing (which cannot affect the outcome of the transaction).

4.3.4 Transaction Resources

Transaction resources are provided by transactional resource managers and MUST implement the javax.transaction.xa.XAResource interface described in [6]. An XAResource object can be enlisted with the transaction to ensure that work is performed within the scope of the transaction. The XAResource interface is driven by the transaction manager during the completion of the transaction and is used to direct the resource manager to commit or rollback any changes made under the transaction.

4.4 Locating OSGi transaction services

The Java EE specifications define standard JNDI names for the UserTransaction and TransactionSynchronizationRegistry interfaces in a Java EE server environment and deliberately do not define a standard means for acquiring an implementation of the TransactionManager interface. This is because the latter is considered to be a part of the internal implementation of a Java EE application server. An OSGi transaction service implementation MUST register service objects for the UserTransaction, TransactionSynchronizationRegistry and TransactionManager interfaces in the OSGi service registry, using the names “javax.transaction.UserTransaction”, “javax.transaction.TransactionSynchronizationRegistry” and “javax.transaction.TransactionManager” respectively. An OSGi transaction service MAY put restrictions on which bundles can use these service objects as described in 5.1.

An OSGi transaction service implementation MAY also bind references to UserTransaction and TransactionSynchronizationRegistry in a JNDI namespace.

4.5 Use Cases

4.5.1 Create Transaction

An application component uses the UserTransaction service interface to start a new transaction. If there is already active transaction in the context of the current thread, the transaction manager will indicate an error.

4.5.2 Join Transaction

When a transaction is started, the application performs some operations on the system. While modifying the current state, it invokes some methods or other services. These services are resource managers that can participate in the transaction. The resource managers each provide an XAResource object and join the transaction associated with the current thread via the enlistResource(XAResource) method of the Transaction object.

4.5.3 Commit Transaction

After performing the required operations, the transaction originator decides whether to initiate commit or rollback processing and requests it via the UserTransaction interface. During commit processing if at least one of the resources participating in the transaction fails to perform the required operations (vetoes the prepare phase), the transaction is rolled back.
4.5.4 Prepare Resource
The transactional manager uses the “two phase commit” protocol to ensure the consistent outcome across all resource managers. When the originator requests a commit of the transaction, the TM calls “prepare” on each participating resource. In this stage, the resource provisionally performs any updates and decides whether it can honour a commit outcome if that is what the external coordinator decides. It then waits to be told the final commit or rollback decision.

4.5.5 Commit Resource
If all participating resources have been successfully prepared, the TM calls the commit() method on each resource manager. The resource manager is responsible for making changes to the transactional resource persistent and visible outside the transaction.

4.5.6 Rollback Transaction
If the originator decides to request a rollback of the transaction, or if the transaction fails before a completion request is made, the original state of the system is restored to what existed before the transaction was started. Rollback might be called either because commit failed or for some external reasons – like operation timeout as example.

4.5.7 Rollback Resource
During a transaction rollback, the TM calls “rollback” on each resource. This method discards any provisional updates within the transaction and so restores the original state of the resource.

4.5.8 Assign Transaction to Thread
A transaction MUST NOT be associated with more than one thread at a time but MAY be moved over time from one thread. While transaction-thread association is provided by the transaction manager, any movement of the transaction from one thread to another – via the suspend/resume methods of the TransactionManager interface - is driven by the framework hosting the OSGi transaction service and it is the responsibility of that framework and the transactional resource managers to understand which transaction context the transactional resources are running under.

4.6 Functionality

4.6.1 Scope of a global transaction
A transaction context is started and ended by requests from a transaction originator. In between, a transaction manager manages the transaction. Transactional resources may be enlisted in the transaction during its lifetime. Those transactional resources are coordinated to an atomic outcome by the transaction manager at the end of the transaction.

4.6.2 Correctness of the State
At the end of transaction, the application must decide whether the changes should be made persistent (committed) or rolled back. The application requests the transaction manager to perform commit or rollback processing. The collection of state changes to all transactional resources made under the transaction can have ACID properties when a global transaction is used. The transaction manager itself is responsible for providing the Atomic property of ACID by driving all resource managers to the same outcome. The resource managers are responsible for the Isolation and Durability properties of the changes. The application and the resource managers provide the Consistency of the data changes.
4.6.3 End of Transaction

The transaction is disposed after it has been successfully committed or rolled back. The transaction manager automatically disassociates the transaction from the participating thread. This allows a new transaction to be started for that thread.

4.6.4 Performance

Global transaction processing can be expensive in terms of performance and resource utilization. Therefore, to optimize performance you may choose to execute a majority of the code without a transaction, and use transactions only when necessary. Using the credit card processing example, you may not use transactions to do *data loading, validation, verification, and posting*. However, at the point when you transfer money from the account holder to the holding bank you would then start a transaction. The XA and JTA specifications provide opportunities for implementation optimizations such as the well-known “one phase commit optimization of two-phase commit” which causes almost all of the cost of 2PC to be realized only when a transaction with more than one transactional resource begins commit processing.

4.6.5 Management of Transaction

The component that completes a transaction should be the same component that originated it. Therefore, only the business method that started the transaction should invoke the commit() and rollback() methods. Spreading transaction management throughout the application adds complexity and reduced maintainability of the application from a transaction management standpoint.

4.6.6 Heuristic Exceptions

Heuristic outcomes can result if a transactional resource does not keep the promise it made during the prepare phase, most typically as a result of a database administrator forcing a unilateral and administrative outcome for operational reasons. Under such circumstances, the administrator may need to take further action to maintain integrity across the global transaction as a whole.

4.6.7 Examples

4.6.7.1 Example 1 - Creating and using a Transaction

The following pseudo code illustrates the creation and use of a transaction involving a transactional resource "ConfigResource":

```java
ServiceReference txRef = 
    bundleContext.getServiceReference("javax.transaction.UserTransaction");
UserTransaction tx = (UserTransaction)bundleContext.getService(txRef);

// begin transaction
try {
    tx.begin();

    // perform some operations in the context of the transaction
    try {
        // Create a transactional resource
        ConfigResource config = ...;
        // Perform some transactional work
        Configuration x = config.createConfiguration("abc");
        tx.commit(); // make changes persistent
    } catch (Throwable th) {
        th.printStackTrace();
        tx.rollback(); // rollback changes on fail
    }
```
4.6.7.2 Example 1 - (Resource) Participating in Transaction

The following pseudo code illustrates the enlistment of an XAResource with a transaction; not all the methods of the XAResource interface are shown.

```java
class ConfigResource implements XAResource {

    public Configuration createConfiguration(String pid) {
        // TransactionManager can be obtained from service registry
        ServiceReference tmRef =
            bundleContext.getServiceReference("javax.transaction.TransactionManager");
        TransactionManager tm = (TransactionManager)bundleContext.getService(tmRef);
        // enlist the transactional resource
        tm.getTransaction().enlistResource(this);
        // Transactional operation
        addLog("createConfiguration", pid);
        ...
    }

    public void prepare(Xid xid) {
        // Make and persist a provisional “after copy” of data for this xid
        ...
    }

    public void commit(Xid xid, boolean onePhase) {
        // Replace the “before” copy with the “after” copy
        // Forget the transaction
    }

    public void rollback(Xid xid) {
        // Discard any provisional “after” copy
        // Forget the transaction
    }

    ...
}
```

5 Security Considerations

5.1 ServicePermissions

An OSGi service framework provider MAY chose to restrict which bundles are allowed to register OSGi transaction services. A framework provider optionally does this by requiring that any registering bundle has the ServicePermission.REGISTER permission for each of the service names listed in 4.4 that it wishes to register.
Similarly, an OSGi service framework provider MAY chose to restrict which bundles are allowed to get references to OSGi transaction service objects. A framework provider optionally does this by requiring that any bundle requesting a service reference has the ServicePermission.GET permission for the service names listed in 4.4 that it wishes to get.

6 Document Support

6.1 References


6.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Peter Kriens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>aQute</td>
</tr>
<tr>
<td>Address</td>
<td>9C, Avenue St. Drézéry</td>
</tr>
<tr>
<td>Voice</td>
<td>+15123514821</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:Peter.kriens@aQute.biz">Peter.kriens@aQute.biz</a></td>
</tr>
</tbody>
</table>
### 6.3 Acronyms and Abbreviations

### 6.4 End of Document
Abstract

This RFC contains a design that meets the requirements described in RFPs 79 and 88. The solution defines a minimal level of feature/function for distributed OSGi processing, including service discovery and access to and from external environments. This solution is not intended to preclude any other solution and is not intended as an alternative to Java EE, SCA, JBI, or any other external API set that may be mapped onto OSGi.
5.2.2 Interface description

5.2.2.1 Distribution Software Interface

5.2.2.2 Exception Handling

5.3 Discovery Service

5.3.1 Functionality

5.3.2 Interface description

5.3.2.1 Discovery interface

5.3.2.2 ServicePublication Interface

5.3.2.3 DiscoveredServiceTracker interface

5.3.2.4 DiscoveredServiceNotification interface

5.3.2.5 ServiceEndpointDescription interface

5.3.3 Discovery using a local file(s)

5.4 Service Registry Hooks

5.4.1 Registration of Remote Services in Local Service Registry

5.4.2 Additional filtering

5.5 Service Programming Model

5.5.1 Service interface description

5.5.2 Properties

5.5.2.1 Definition of new Properties

5.5.2.2 Standard Properties

5.5.3 Intents

5.5.3.1 Example of using Intents

5.5.3.2 Defining Intents

5.5.3.3 OSGi-defined Intents

5.5.3.4 SCA-defined Intents

5.5.3.5 Security Intents

5.5.3.6 Reliable Messaging Intents

5.5.3.7 Transactional Intents

5.5.3.8 Miscellaneous Intents

5.5.3.9 Not Applicable Intents

5.5.3.10 Qualified Intents

5.5.3.11 Publishing of Qualified Intents

5.5.3.12 Profile Intents

5.5.3.13 Publishing of Profile Intents

5.5.3.14 Exclusive Intents

5.5.4 Configuration type

5.5.5 Service Factories

5.6 Collaboration of new and changed entities

5.6.1 Interactions on the service provider side

5.6.1.1 Exposing a Service remotely

5.6.1.2 Modification of service properties

5.6.1.3 Service Unregistration

5.6.2 Interactions on the service consumer side

5.6.2.1 Lookup for a remote Service

5.6.2.2 Service invocation

5.6.2.3 Service Unregistration

5.6.3 Interactions with Non-OSGi service providers and consumers

5.6.4 Lifecycle dynamics

5.7 Service Distribution using SCA Metadata

5.7.1 Bindings

5.7.1.1 Bindings on Services

5.7.1.2 Bindings in Service Descriptions
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>
<tbody>
<tr>
<td>Initial</td>
<td>Jul 17, 2007</td>
<td>Initial draft with information based on RFP 88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eric Newcomer, IONA, <a href="mailto:eric.newcomer@iona.com">eric.newcomer@iona.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>David Bosschaert, IONA, <a href="mailto:davidb@iona.com">davidb@iona.com</a></td>
</tr>
<tr>
<td>0.1</td>
<td>July 27, 2007</td>
<td>Added parts related to RFP 79 (service discovery)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tim Diekmann, Siemens Communications, <a href="mailto:tim.diekmann@siemens.com">tim.diekmann@siemens.com</a></td>
</tr>
<tr>
<td>Revision</td>
<td>Date</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0.2</td>
<td>August 1-8</td>
<td>Added introductory information, incorporated edits, and filled in sections from relevant RFPs.</td>
</tr>
<tr>
<td>0.3</td>
<td>September, 2007</td>
<td>Changes following feedback from the August Face-to-Face &lt;br&gt;Tim Diekmann, Siemens Communications, <a href="mailto:tim.diekmann@siemens.com">tim.diekmann@siemens.com</a> &lt;br&gt;Eric Newcomer, IONA, <a href="mailto:eric.newcomer@iona.com">eric.newcomer@iona.com</a> &lt;br&gt;David Bosschaert, IONA, <a href="mailto:davidb@iona.com">davidb@iona.com</a></td>
</tr>
<tr>
<td>0.4</td>
<td>October, 2007</td>
<td>Tim Diekmann, Siemens Communications, Philipp Konradi, Siemens Corporate Technologies</td>
</tr>
<tr>
<td>0.5</td>
<td>November, 2007</td>
<td>Tim Diekmann, Siemens Communications, Philipp Konradi, Siemens Corporate Technologies &lt;br&gt;Klaus Kunte, Siemens Enterprise Networks GmbH &amp; Co KG</td>
</tr>
<tr>
<td>0.6</td>
<td>February, 2008</td>
<td>Changes as a result of discussions at January F2F &lt;br&gt;Deleted comments that seemed resolved or discussed. &lt;br&gt;Eric Newcomer</td>
</tr>
<tr>
<td>0.7</td>
<td>March, 2008</td>
<td>Tim Diekmann, Siemens Communications. Accepted all changes.</td>
</tr>
<tr>
<td>0.8</td>
<td>May, 2008</td>
<td>Eric Newcomer, accepted changes, incorporated text about intents, cleaned up comments following their resolution. &lt;br&gt;Graham Charters &amp; Philipp Konradi, qualified intents section.</td>
</tr>
<tr>
<td>0.9</td>
<td>June, July 2008</td>
<td>Eric Newcomer, further editorial cleanup.</td>
</tr>
<tr>
<td>0.9.1</td>
<td>July 2008</td>
<td>Tim Diekmann, minor editorial changes, bug 719</td>
</tr>
<tr>
<td>0.9.2</td>
<td>July 2008</td>
<td>Eric Newcomer, changes from July F2F &amp; bug list</td>
</tr>
<tr>
<td>0.9.3</td>
<td>August 2008</td>
<td>David Bosschaert, changes relating to bugs 689, 729 and 735.</td>
</tr>
<tr>
<td>0.9.4</td>
<td>August 2008</td>
<td>Graham Charters, changes relating to bugs 606, 685, 755, 756, 759</td>
</tr>
<tr>
<td>0.9.5</td>
<td>August 2008</td>
<td>Viktor Ransmayr &amp; Philipp Konradi, update interface description for Discovery Service</td>
</tr>
<tr>
<td>0.9.6</td>
<td>September 2008</td>
<td>Graham Charters, minor updates to SCA section based on 119 authors call.</td>
</tr>
<tr>
<td>0.9.7</td>
<td>September 2008</td>
<td>David Bosschaert: Changes as outlined in the resolution to bug 730.</td>
</tr>
<tr>
<td>0.9.8</td>
<td>September 2008</td>
<td>Graham Charters: changes for bugs 606 (configuration.type), 685 (SCA section) and 725 (Intents schema)</td>
</tr>
<tr>
<td>0.9.9</td>
<td>October 2008</td>
<td>David Bosschaert: Changes relating to bugs 740, 856, 782, 808, 731</td>
</tr>
<tr>
<td>0.9.9.1</td>
<td>November 2008</td>
<td>Graham Charters: Changes for bugs 868 (SCA intent definitions) and 697 (best practices – remote vs local)</td>
</tr>
<tr>
<td>Revision</td>
<td>Date</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0.9.9.2</td>
<td>November 2008</td>
<td>Added response descriptions for messaging intents. Added &lt;service-descriptions/&gt; schema (bug 757) and updated associated example.</td>
</tr>
<tr>
<td>0.9.9.3</td>
<td>December 2008</td>
<td>Philipp Konradi: resolution for bug 761 (Discovery auto-publish design)</td>
</tr>
<tr>
<td>0.9.9.4</td>
<td>December 2008</td>
<td>David Bosschaert: resolution for bug 735 (removal of getPublishedServices() API from DistributionProvider service).</td>
</tr>
<tr>
<td>0.9.9.5</td>
<td>December 2008</td>
<td>Tim Diekmann (<a href="mailto:tdiekman@tibco.com">tdiekman@tibco.com</a>): preparation for vote in EEG</td>
</tr>
<tr>
<td>0.9.9.6</td>
<td>December 2008</td>
<td>Graham Charters: bugs 853 (scenario clarification) and 758 (passByValue as the default semantic).</td>
</tr>
<tr>
<td>0.9.9.7</td>
<td>December 2008</td>
<td>David Bosschaert, rewrote section 5.3.2 (Discovery using local files) as discussed during the authors call.</td>
</tr>
<tr>
<td>0.9.9.8</td>
<td>December 2008</td>
<td>Philipp Konradi: updated Discovery service interface, updated section 5.6 (Collaboration of new and changed entities), deleted optional Discovery properties, moved chapter “Discovery Service federation and interworking” to Best Practices</td>
</tr>
<tr>
<td>0.9.9.9</td>
<td>December 2008</td>
<td>Philipp Konradi: changed white-space separated propertyvalues to String[], updated description of osgi.remote.configuration.type to contain multiple values, other minor clarifications in preparation for vote.</td>
</tr>
<tr>
<td>1.0</td>
<td>December 2008</td>
<td>Final version out for vote.</td>
</tr>
<tr>
<td>1.0.1</td>
<td>February 2009</td>
<td>Tim Diekmann, minor changes, updated osgi.intents to service.intents according to bug 807</td>
</tr>
</tbody>
</table>

# 1 Introduction

This RFC is being created as a design document to meet the requirements described in RFPs 79 and 88. The focus is on defining a possible solution within the OSGi environment to provide a minimal level of feature/function for distributed OSGi processing, including service discovery and access to and from external environments. This solution is not intended to preclude any other solution and is not intended as an alternative to JEE, SCA, JBI, or any other external API set that may be mapped onto OSGi, although the solution is intended to enable interworking with external implementations of those and other technologies.

The solution is intended to allow a minimal set of distributed computing functionality to be used by OSGi developers without having to learn additional APIs and concepts. In other words, if developers are familiar with the OSGi programming model then they should be able to use the features and functions described in this solution very naturally and straightforwardly to configure a distribution software solution into an OSGi environment to meet requirements stated in RFPs 79 and 88. If developers need to use advanced distributed
computing capabilities they can use any other supported APIs defined for OSGi deployment to augment or replace the basic functionality described in this RFC.

This RFC is based on describing the minimal extensions necessary to the existing OSGi environment for the purposes of allowing:

- An OSGi bundle deployed in a JVM to invoke a service in another JVM, potentially on a remote computer accessed via a network protocol
- An OSGi bundle deployed in a JVM to invoke a service (or object, procedure, etc.) in another address space, potentially on a remote computer, in a non OSGi environment
- An OSGi service deployed in another JVM, potentially on a remote computer, to find and access a service running in the “local” OSGi JVM (i.e. an OSGi deployment can accept service invocations from remote OSGi bundle
- A program deployed in a non OSGi environment to find and access a service running in the “local” OSGi JVM (i.e. an OSGi deployment can accept service invocations from external environments)

Basic assumptions include that the default mode of distributed access is consistent with the current OSGi programming model (i.e. a service oriented request/response model) and that in most cases the use of distribution software can be accomplished through the use of configuration and deployment metadata. The configuration and deployment metadata is based on the Service Component Architecture (SCA) intent model of abstracting distributed computing capabilities. The design is intended to work with any broadly adopted type of distributed computing software system, such as Web services, CORBA, or messaging.

Existing distributed computing technologies are used in all cases to meet the requirements. A further distinction is drawn between solutions that use the same distributed software system for all communications, and solutions that use multiple distributed software systems. When multiple distributed software systems are involved additional metadata may be required to ensure consistency and compatibility of the configurations.

This RFC does not define any new distributed communication protocols, data formats, or policies: it simply defines an extension to the OSGi programming model and metadata that defines how to access and load modules for existing communication protocols, data formats, and policies (i.e. qualities of service assertions and associated configurations) to meet the requirements of RFPs 79 and 88.

### 1.1 Terminology

**OSGi service platform**: See OSGi core specification chapter 1.

**OSGi bundle**: See OSGi core specification chapter 3 and 4.

**OSGi service**: See OSGi core specification chapter 5.

**OSGi service registry**: See OSGi core specification chapter 5.

**Component**: A piece of code (e.g. similar to a Spring bean or a POJO) that is packaged and deployed in a bundle.

**Application**: A set of bundles that are logically coupled to perform a common task. The bundles of this application don’t have to be deployed in the same service platform, but can be spread over multiple service platforms.
**Distribution software (DSW):** A software entity providing functionality to an OSGi service platform that supports the binding and injection of services in other address spaces or across machine boundaries, using various existing software systems.

**Discovery service:** A software entity providing functionality to an OSGi service platform that supports the publishing and lookup of services in other address spaces or across machine boundaries, using various existing discovery systems.

**Service consumer:** A bundle which requires a service from other service platforms.

**Service provider:** A bundle which provides a service to other service platforms.
### 1.2 List of Symbols

The following symbols are used in the drawings in this document to illustrate the desired behavior of the distributed OSGi design.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
</table>
| ![OSGi Service](symbol) | OSGi Service | - can be registered by bundles (register)  
- can be looked for and used from other bundles (get)  
- can be listened on the service listener, e.g. a service tracker listens on service events (listen)  
- can be hooked into the process of service registration and lookup (hook)  
- can be configured to be accessed remotely |
| ![Bundle](symbol) | OSGi Bundle | - provides modularization and encapsulation of components  
- is a deployment unit (software provisioning) for the OSGi runtime |
| ![Extender](symbol) | Extender | An (extender) bundle listens for life-cycle events of other bundles and synchronously acts if necessary e.g. to inject dependencies. The extender bundle is the one the arrow starts at. |
| ![External Interface](symbol) | External Interface | - provides an interface outside of a local OSGi Service Platform  
- exposing transport or communication protocols, e.g. SOAP/HTTP, CORBA/IOP, RMI, etc. |
| ![Platform](symbol) | Non OSGi Platform | - provides a component based environment for enterprise applications  
- offering non OSGi technologies, e.g. SCA, Spring, etc. |
| ![Platform](symbol) | OSGi Service Platform | - provides a service-oriented, component-based environment  
- focused on the component integration and the software lifecycle |

Also UML notation was used for some diagrams in this document. Please refer to [www.uml.org](http://www.uml.org) for details on the notation.

### 2 Application Domain

[copied and combined from RFPs 79 & 88]

The primary design addressed by this RFC is intended to meet requirements for the heterogeneous enterprise IT environment that includes existing and new non-OSGi based applications that need to
communicate with OSGi based applications, and with which OSGi based applications need to communicate, including connecting embedded systems to enterprise systems.

Examples of such applications include internet banking applications connected to mainframe databases, travel applications with multiple providers of travel item reservations that all use different technologies, telecommunications industry services for broadband telephony and television that rely on legacy billing applications, and so on. Typical enterprise deployments include large scale applications, which require high availability, reliability, and scalability of the provided services.

Standalone or single technology applications (i.e. OSGi only) are also in scope, because of the fact that OSGi based applications might be deployed in more than just one OSGi platform and for scalability and availability purposes need to be able to find each other across the platform boundaries.

Some core features of heterogeneous enterprises:

- “Stove-piped” applications written using different languages and software systems, including but not limited to .NET, JEE, C++, CORBA, message oriented middleware, TP monitors, data base management systems, packaged applications, EDI, and Web technologies
- Applications built and maintained by separate departments and business divisions that were not designed or built using any consistent principles, and may or may not have integration points exposed.
- Multiple communication protocols and paradigms (i.e. synch and asynch) for interacting with different applications
- Multiple data formats for the same, or similar data items that need to be accessed consistently or reconciled for both read and update operations.
- Quality of service requirements inherent in existing applications, including security, transactionality, reliability, and performance service levels of agreement that need to be met, sometimes expressed in machine readable policies and configuration files

While OSGi has some of the capabilities in place for interaction with external systems, the requirements of interacting with heterogeneous IT environments is often dictated by the requirements of the existing applications, since they represent communications protocols, data formats, programming languages, software systems, and qualities of service agreements already in place for the business.

3 Problem Description

3.1 From RFPs 79 & 88:

Sometimes the objective of the interaction between new OSGi based applications and existing applications will be to perform retrieval and update operations directly on existing data resources. Other times the objective of the interaction will be to use an existing or new program to serve as a proxy or intermediary for another program’s data operations. Other times the objective of an interaction will be to request the execution of some business logic, or to evaluate some data, or perform a complex calculation and return the results.

Independent of the interaction scenario, the services of the new OSGi based application need to be discovered by potential clients running outside of the hosting OSGi platform.

The problem space, therefore, has the following characteristics:
• Local OSGi services are only accessible from inside the same OSGi platform execution environment.

• Remote OSGi services need to be discovered and accessible from outside of the OSGi platform execution environment.

• Information about the distributed capability needs to be included in the registration and discovery of remote OSGi services. A mechanism needs to be defined for plugging in or binding to different communications protocols and data formats. A mechanism needs to be defined for plugging in or binding to different data formats – the requirement here in both cases can also be stated as how to bind an OSGi service to a transport layer and (potentially separately) a data format layer.

• A mechanism that defines how to mix ‘n’ match communications protocols and data formats so that data formats can be reused over multiple transports (e.g., allow SOAP over JMS or binary over HTTP).

• Existing legacy systems need to be able to locate and access OSGi services of new applications.

• Embedded applications need to interoperate with enterprise applications.

• Besides the pure interface definition additional metadata needs to be available about the services that are found remotely in order to assess their eligibility for reference binding. This metadata is part of the service contract and may include non-functional requirements.

• A mechanism to download a remote service.

• A mechanism to configure or plug in quality of service capabilities.

• A mechanism to interact with external (i.e. remote) data resources.

The requirements of interacting with existing heterogeneous IT environments is often dictated by the requirements of the existing applications, since they represent communications protocols, data formats, programming languages, software systems, and qualities of service agreements already in place for the business.

Another requirement centers on interoperability:

1. A service published remotely through OSGi implementation A should be accessible from another Service that runs in OSGi implementation B.

2. Implementations A and B could be based on entirely different OSGi runtimes.

3. For a user of the OSGi runtime, it should be easy to identify that a certain OSGi runtime can interoperate with another OSGi runtime by examining the service properties and any associated metadata. So let’s say the user already has an OSGi runtime that exposes its services using a certain wire protocol, e.g. SOAP/HTTP. If the user starts using another OSGi runtime that says that it supports SOAP/HTTP they should interoperate.

Whether or not an IDL or some other formalism like special use of Java Interfaces would be needed to satisfy this is certainly a valid discussion point, but it would be good to try and solve it within the boundaries of Java Interfaces, simply because this concept is already used in OSGi.

### 3.2 Scenario diagrams

Schematically, the problem domain centers on a solution to the following scenarios. Note that the non-OSGi clients and servers mentioned may represent existing and legacy applications that typically can’t be modified.

The scenario illustrated in Figure 1 focuses on a client in one OSGi platform that needs to invoke on a Service that lives in another OSGi platform. Both client and server might initially not be written to be distributed. However, it may in certain cases be a possibility to tweak the client and/or service code to make them behave better in this distributed scenario.
Note that in this case an implementation might choose to use an optimized protocol to communicate between the OSGi runtimes. Note also, that if the same distribution software (e.g. ESB) is used in both service platform instances, then the configuration required can also be optimized.

![Diagram of OSGi service consumer using a remote OSGi-service provider](image)

**Figure 1 OSGi service consumer using a remote OSGi-service provider**

The general use case of distributed OSGi is depicted in Figure 1. A client hosted in OSGi Service Platform A wants to use a service provided by another bundle hosted in OSGi Service Platform B. Since this is a remote service invocation spanning multiple framework processes (i.e., multiple JVMs), some intermediary bundle is required in both service platforms to marshal and unmarshal the communication objects. This RFC describes the mechanisms how to find and match client and server as well as how to implement the intermediary bundle to enable the remote invocations.

It is the intent of this RFC to allow for any implementation of the distribution software as shown in the picture utilizing any protocol for the communication, associating metadata with the service to indicate that it's remotable, and with which distributed software characteristics (as expressed using "intents").

Note: As described in the requirements section, RFC 119 is also addressing the scenarios in which the client side is hosted in a non-OSGi environment. In this case, the left side would be replaced by another client hosting platform, e.g., .NET. Additionally, OSGi based clients should be able to remotely access services hosted in a non-OSGi environment, which would mean that the right side is replaced with a different hosting platform.

### 3.2.1 Consumer Side

The following diagram illustrates the detailed scenario from the consumer side.
Figure 2 Service Consumer - in OSGi framework

Figure 2 shows the OSGi implementation in the client OSGi platform A. Bundle A is interested in Service B and performs a lookup in the service registry (expressing the metadata intents it requires, if any (see Section 5.5.3 for the definition of intents)) or uses a ServiceTracker to listen for events regarding Service B – step (1a). Service B can have metadata properties associated with it to indicate that it’s accessible remotely. Optionally, a dependency management mechanism such as Declarative Services or Blueprint Service based components (see RFC 124), or others could perform the dependency checking and register such a listener (1b).

Step (2) in the diagram refers to RFC 126, service registry hooks (see also Section 5.4). It allows the distribution software to register a hook in the service registry, which is called when a service is being looked-up or requested.

In the optional step (3a), the distribution software could use the discovery service to perform the lookup of Service B over the network. The Discovery service is an optional service and registers its service upon startup. Discovery allows for synchronous as well as asynchronous discovery of services suitable for providing Interface B and meeting the requirements of Bundle A. Step (3b) illustrates the direct use of the discovery protocol, while step (3c) illustrates the use of a mechanism that tracks previously discovered services.

Step (3) is optional, because the distribution software may also acquire the information about Service B through other means, such as static configuration (wiring) as part of its implementation, or using a local file (see Section 5.3.2).

The distribution software and the Discovery service do not have to come from the same vendor and adhering to the OSGi specification allows for seamless integration of different discovery and distribution implementations.

In step (4) the distribution software creates a local endpoint for the discovered provider of Service B. The deployed protocol depends on the available protocols for Service B. (See discussion below for details about the provider side.)

In step (5), the distribution software registers the proxy with Interface B, which causes in step (6a) and (6b) the service reference to be returned to the calling party or injected by the dependency mechanism.
3.2.2 Provider Side

The following diagram illustrates the provider side.

![Diagram showing the provider side of OSGi framework]

Figure 3: Service provider - OSGi framework

In Figure 3 it is shown how Bundle B inside the OSGi Service Platform B registers a Service B in step (1a), including its metadata stating it’s remotely accessible and with which characteristics (i.e. any specified properties and intents). Optionally, this step could also be performed by a dependency management mechanism such as Declarative Services, Blueprint Services, or any other non-standard implementation (1b).

In step (2) the distribution software is notified about the registration of Service B and using additional information provided by step (3) in which the extender model can obtain any intents). This could be done by an extender or through properties as part of the registration of Service B. To make Service B reachable through a communication protocol the distribution software creates a local endpoint for the supported protocol(s) in step (4).

The OSGi Service Platform B may optionally also have deployed a discovery bundle as specified in this RFC. The discovery bundle registers its standard interface in step (5) and the distribution software is notified about the presence of the discovery service in step (6). Using the discovery service, the distribution software may then publish the information about the availability of Service B using any discovery protocol that the discovery service supports.

Note: It is entirely possible and encouraged that there are 0..n different discovery bundles deployed in the OSGi service platform. Multiple distribution software system types are also permitted.

3.2.3 A non-OSGi distributed client using an OSGi service

Figure 4 shows a legacy client that needs to invoke a service provided by an OSGi service. The client is written in a programming language such as C++ and uses a certain distributed protocol, such as SOAP/HTTP(S), CORBA/IIOP or RMI to access this service.
Figure 4 Remote non-OSGi service consumer using an OSGi service provider

As illustrated in Figure 4, a C++ client deployed in a runtime environment external to OSGi accesses a service deployed in an OSGi platform using one of the communication protocols supported by a DSW deployed in the OSGi platform.

Figure 5 illustrates additional detail of the OSGi runtime part of this scenario. While certain services could be distributed, and are therefore marked with the publish metadata property, it is also possible for other services to exist in the same OSGi platform that aren’t distributed. A co-located client would be capable of making a direct invocation on services that are in the same OSGi platform regardless of their distribution status. Note, in doing this, care must be taken to ensure that the possible change in call semantics (e.g. from remote pass-by-value to local pass-by-reference in most cases) does not adversely alter the behavior of the service.

Figure 5 Service consumer uses a distributed but locally available service provider

Figure 5 illustrates the scenario in which a remote service call from an external environment passes the call to a local OSGi service to invoke the actual service target from the remote invocation (and the actual service target could itself be in a remote OSGi platform).

3.2.4 An OSGi client using a remote non-OSGi Service

Figure 6 illustrates how an OSGi client invokes a legacy service. The service is exposed using a particular type of middleware, e.g. SOAP/HTTP, CORBA/IIOP, RMI, etc. The service is also identified within the OSGi environment as remotely accessible using OSGi metadata (i.e. properties and intents).
Figure 6 OSGi service consumer using a remote non-OSGi-service provider

Figure 6 shows how an OSGi service client can access a service proxy created using a distributed software system that connects remotely to a C++ server deployed in an external runtime environment, using a distributed communications protocol and data format supplied by the DSW configured into the OSGi platform.

3.3 Roles

When creating a distributed application people with different roles are involved, as shown in Figure 7. This section describes the roles relevant to this document.

The following table shows how these roles collaborate with each other, which artifacts are required to perform the tasks in these roles, and which artifacts are produced:

<table>
<thead>
<tr>
<th>Role</th>
<th>Required Artifacts</th>
<th>Performed Tasks</th>
<th>Collaborates With</th>
<th>Produced Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution Architect</td>
<td>Application Requirements</td>
<td>Analyses requirements</td>
<td>Solution Deployer</td>
<td>Solution Specification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defines SOA architecture</td>
<td>Component Designer</td>
<td>Component</td>
</tr>
</tbody>
</table>
Defines the Service Interface and some properties

Requirements
Service Interface Definition
(e.g. UML class diagram, etc.) and property definition (e.g. remoteable)

<table>
<thead>
<tr>
<th>Component Designer</th>
<th>Component Specification</th>
<th>Solution Architect</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Requirements</td>
<td>Service Interface Specification</td>
<td>Solution Architect Developer Assembler</td>
<td></td>
</tr>
<tr>
<td>Service Interface Definition and property definition</td>
<td>Designs the service implementation</td>
<td>Solves the service interface</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specifies the service interface</td>
<td>Defines service-specific properties, intents and optional metadata</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defines service-specific properties, intents and optional metadata</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Developer</th>
<th>Component Designer</th>
<th>Assembler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Specification</td>
<td>Solution Architect Developer</td>
<td></td>
</tr>
<tr>
<td>Service Interface Specification</td>
<td>Solution Architect Developer</td>
<td></td>
</tr>
<tr>
<td>Definition of Service Properties, intents and optional metadata</td>
<td>Solution Architect Developer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler</th>
<th>Solution Deployer</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>Bundles</td>
<td></td>
</tr>
<tr>
<td>Definition of Service Properties, intents and optional metadata</td>
<td>Service Properties, intents and optional metadata (i.e. bundle specific property file or XML file containing defaults)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution Deployer</th>
<th>Solution Architect</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution Specification</td>
<td>Bundles</td>
<td></td>
</tr>
<tr>
<td>Bundles</td>
<td>Service Properties, intents and optional metadata</td>
<td></td>
</tr>
<tr>
<td>Service Properties, intents and optional metadata</td>
<td>Application Configuration Properties, intents and optional metadata (e.g. communication protocols, policies, etc.)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: within a role the members collaborate with each other, e.g. the component designer collaborates with other component designers

### 3.3.1 Solution Architect

The Solution Architect is responsible for defining the functional and non-functional component requirements and for providing the service interface definition. In addition he provides the solution specification to the solution deployer.

He analyses the application requirements and models an appropriate SOA architecture in which functionality is decomposed into services, which can be distributed over a network and can be combined together and reused to create applications.

The solution architect divides the required functions of the application between the components and specifies this architecture design in the component requirements in an informal manner. Furthermore he provides the service interface definitions in form of UML diagrams, which hides the concrete technology used for the service interface like WSDL for Web services, CORBA IDL, RPC IDL, etc.
3.3.2 Component Designer

The component designer is responsible for creating the service interface and for providing the component specification, which specifies the design of the service implementation. In addition he provides the service-specific properties, intents and optional metadata to the assembler and developer.

He works from the service interface definition provided by the solution architect, models the interface objects, e.g. request/response objects, data types, exceptions, etc, and maybe with support of a tool he creates the service interface, e.g. WSDL file, IDL file, etc.

The Component Designer also works with definitions of remote services to be consumed by the component. These are provided by the solution architect and will consist of a service interface description (e.g. Java, WSDL, IDL), and optionally service properties and intents. These remote service definitions may be defined by the solution, or could be external interfaces dictated by a third-party.

Analyzing the component requirements the component designer specifies the design of the service implementation in the component specification. He defines the service-specific properties, intents and optional metadata inherently associated with the component, such as ‘osgi.remote.interfaces’, call by reference / call by value semantics, required QoS, etc. He may define additional properties which depend on the concrete environment the component is used in and thus needs to be provided at runtime.

3.3.3 Developer

The developer is responsible for building the components, which is a set of classes comprising the business logic implementation, according to the component specification.

He works from the service interface provided by the component designer, codes the business logic and creates the business data objects.

3.3.4 Assembler

The assembler is responsible for assembling the components into bundles, which are installable packages, and for providing the defaults for service properties, intents and optional metadata.

He collects and validates the produced components and packages them appropriately together for an OSGi bundle, e.g. by analyzing which services communicate with each other he decides to package bundles for service consumers and service providers.

Based on the service property, intent and optional metadata definitions from the component designer and the developer the assembler creates the service properties, intents and optional metadata by means of types and defaults, and provides additional bundle specific properties, intents and optional metadata, e.g. reuse in multiple applications, etc. These properties are supplied either in a property file, which are configuration values managed as key/value pairs, or in XML files, specifying intents, bindings, policy sets and properties. (Depending on the property file versus XML file discussion)

3.3.5 Solution Deployer

The solution deployer is responsible for deploying the application, which is a set of bundles that are coupled together to perform the solution, and for providing distribution configuration.

He collects the OSGi bundles from the assemblers and configures the distribution software to distribute the services by providing configuration properties, intents and optional metadata as required by the solution specification, e.g. communication protocols to be used, policies which needs to be applied, etc. Additionally he can identify a component as ‘remoteable’ even it was not previously marked as such.

The solution deployer analyses the whole solution for performance issues, and diagnoses errors at the implementation / binding level.

Note: In the end the distribution software is responsible for providing a default for all those properties, intents and metadata that were not set in the steps performed by the previously described roles.
3.3.6 Testing

Testing is part of each role and is accompanied by each produced artifact to ensure performance, robustness and interoperability for the whole solution.

3.3.7 Runtime (Framework)

Controls the lifecycle of services and service dependencies (e.g. DS, Blueprint). Unresolved packages, class loading issues are indicators for improper configuration by the solution deployer.

4 Requirements

4.1 From RFP 79

1. The solution MUST provide means to discover OSGi services from outside the OSGi platform. This includes external clients as well as other OSGi services hosted in separate platforms.
2. The solution MUST support clients independent of the programming language and independent of the location they are at.
3. The solution SHOULD provide means to discover remote services through the local OSGi service registry and standard OSGi mechanisms.
4. The solution MUST be independent of the implementation of the discovery protocol. Multiple implementations must be possible in a single platform. It is understood that only those services will be discoverable that are actually discoverable by the discovery protocol implementation, i.e. a SLP implementation of the service discovery can only discover services that are advertised by SLP.
5. The solution SHOULD avoid or minimize the knowledge about the underlying implementation protocol of the discovery by any service in the local OSGi platform.
6. The OSGi service registry SHOULD contain information about the discovered OSGi services. The information available for the discovery as well as the registration and lookup SHOULD include
   a. Supported communication protocol(s).
   b. Meta-data about the OSGi service, defined by the service itself.
   c. Provided Interface(s)
   d. Quality-of-service information, e.g. transaction support, service specific policies, time constraints, etc.
   e. Transport information, e.g. support for IPv6
   f. Version information of the interface
7. The solution SHOULD support an OSGi service registering multiple interfaces.
8. The solution SHOULD support multiple OSGi services registering the same interface.
9. The solution **MUST** only expose information about those OSGi services that want to be discovered from external clients. Thus, NOT every OSGi service listed in the OSGi registry **MAY** automatically be included in the discovery for external services.

10. The solution **SHOULD** provide for limited visibility of services in the registry based on security mechanisms, e.g. authentication and authorization.

11. The solution **SHOULD** ensure a reasonable response time for service lookup requests.

### 4.2 From RFP 88

1. The solution **MUST** enable interoperability between OSGi developed services (or components) and services (or components) developed using non OSGi environments.

   *The interoperability would typically be provided through the use of existing distributed data bindings and protocols such, e.g. SOAP/HTTP, CORBA/IIOP, JMS, RMI etc. Not all possible integrations need to be delivered, what is needed is an extensible framework that can hold these. The Reference Implementation should come with at least 2 implementations to prove the scenario and pluggability.*

2. The solution **SHOULD** abstract protocols, data formats, and quality of service features in order to be easily adaptable to communication protocols, data formats, and qualities of service found in existing enterprise applications and software systems.

   *This means that the user code should not be required to be written against a particular type of protocol. This should be abstracted.*

3. The solution **SHOULD** be compatible with multiple external programming languages and operating systems.

   *So it should allow interoperability with systems written in a variety of programming languages (e.g. C/C++, .NET, Cobol, scripting languages) running on a number of operating systems such as Windows, UNIX, Mainframes. Note that these external systems do not need to be running an OSGi platform. Interoperability would be provided through the distributed databinding & protocol used.*

4. The solution **SHOULD** be extensible for custom developed interoperability solutions (i.e. users can add their own protocols, data formats, and quality of service extensions).

5. The solution **SHOULD** be configurable and understand policy expressions for the provisioning of the interoperability solutions, especially including the quality of service features.

   *The policy information could be for example expressed as WS-Policy expressions which should give the administrator the ability to define the distribution-related metadata in a declarative way.*

6. The solution **SHOULD** support high availability and performance requirements typical of existing enterprise systems.

7. The solution **SHOULD** bridge the OSGi context sharing mechanism with external context sharing mechanisms (to support stateful failover, shared stateful sessions, etc.).

8. The solution **SHOULD NOT** introduce language specific, protocol specific, or quality of service specific dependencies.

9. The solution for external access **SHOULD** be as consistent as possible with the solution for accessing internal OSGi services, to minimize the amount of effort in moving from one to the other.

10. The solution **SHOULD** provide a consistent mechanism for simultaneously incorporating multiple protocols and data formats.

11. The solution **SHOULD** provide a consistent mechanism for quality of service enhancements.
12. The solution MUST make it possible, but not necessary, for developers to interact with the distributed attributes of the system, such as distributed error conditions, and information around the data binding, transport and QoS.

To give application developers the option to find out the distributed properties of the Service and also be capable of detecting remoting-related specific error conditions if they wish to do so.

13. The solution MUST NOT prevent the use of asynchronous programming models if these are provided by the transport used.

In other words, if the transport provides an asynchronous protocol such as JMS, CORBA one-ways or other message queue or publish-and-subscribe model, it must be possible for the application programmer to take advantage of this asynchronous nature.

14. The solution SHOULD support the capability for a developer to declaratively specify the configuration requirements for a protocol layer.

15. The solution SHOULD allow a deployer to define wiring and configuration information for bundles and create distributed solutions with minimal or no code changes.

4.3 Further requirements

4.3.1 Levels of transparency

While it would nice to be able to turn an existing OSGi Service & Client into a distributed OSGi system without making code changes, it cannot be required that distributed OSGi is entirely transparent. The distributed nature of the system will introduce new scenarios (e.g. new failure scenarios) that were not relevant to non-distributed OSGi. If the program wishes to, it should be allowed to interact with the distributed nature of the system. Therefore, the following levels of transparency should be supported:

1. Completely transparent to the developer. No code changes needed in either Client or Service. Metadata changes will probably be necessary at this level.

2. The programmer should be able to influence the lookup of the Service in the Client based on properties provided in the metadata (e.g. transport, QoS, etc).

3. For any given distributed service it must be possible to find out what the distribution software is and obtain additional metadata that describes the data binding, protocol and QoS.

4. It must be possible to preserve distribution software specific exceptions and handle them as before, if desired. Another exception is defined to indicate a problem occurred in the mapping software.

5 Technical Solution

5.1 Overview of contributions to the OSGi standard

Distributed OSGi enhances the capabilities of the OSGi framework and opens deployment areas in the enterprise market. This section summarizes the changes to the existing specification as of R4.1, and summarizes the additional optional services in distributed OSGi. Subsequent sections provide more details on each.
5.1.1 Summary of Changes to the OSGi Core

The following changes to the core OSGi specification are contained within this design:

- Adaptation of RFC 126 regarding service registry hooks. The proposed solution for this RFC requires the ability to hook into the process of registration, see Section 5.4 for further details.
- Changes to the service programming model for distribution:
  - Reserved properties:
    - service.intents – list of intents satisfied by this service.
    - osgi.remote.interfaces – indicates that the provided service is to be made available remotely, which implies that it is suitable for remote invocations. The value of this property indicates which interface or interfaces implemented by the service are to be exposed remotely.
    - osgi.remote.requires.intents – list of intents that should be satisfied when publishing this service remotely. Provided by the component designer and changeable by the deployer.
    - osgi.remote.configuration.type – identifies the metadata type of additional metadata, if any, associated with the service provider or consumer, e.g. "sca".
    - osgi.remote – indicating that a service implementation is actually remote. This property is set on client side proxies so that the consumer can identify remote services if needed.
  - Metadata for configuring distribution software, which includes basic intents used when there's a single type of distribution software, and additional metadata when multiple types of distribution software are required. This metadata is provided using service properties described above: service.intents, osgi.remote.requires.intents and osgi.remote.configuration.type.

5.1.2 Summary of Additional Services

The distributed OSGi mechanism presented in this RFC 119 is an optional component to an existing OSGi Service Platform as described in the requirements Section 4. As such, the following new OSGi services are proposed to be added to the compendium document of the OSGi specification:

- Distribution software – provides remote invocation capability over one or more protocols; takes care of exposing a service remotely and also provides consumers of remote services with a local reference (proxy) to invoke the remote service. The distribution software will preserve the OSGi service programming model by making OSGi services available to external clients and allowing consumer bundles written in OSGi to bind to external services through OSGi service registry mechanisms. See Section 5.2 for further details.
- Discovery service – an optional service additional to distribution software to locate OSGi based and non-OSGi services over any available network protocol or other means used by the implementation. See Section 5.3 for further details.

5.2 Distribution software

5.2.1 Functionality

The distribution software is responsible for the actual network communication between a remotely available service and its consumer, including the data format (i.e. serialization) and communication protocol.
When a consumer invokes on the remote service the distribution software knows how to marshal the arguments and will then make the dispatch invocation on the remote entity. On completion it will unmarshal the response and return to the caller.

On the provider (service) side, the distribution software knows how to make an OSGi Service available over the network so that it can be invoked remotely. The distribution software may optimize on a particular distributed computing protocol, which may require the OSGi Service Java interface to be mapped onto that technology. Example target technologies include CORBA, RMI and Web Services technologies. However, this specification also allows an implementation to use other protocols and bindings, including proprietary ones.

The distribution software is responsible for interpreting the distribution-related metadata on an OSGi service and making the service available remotely if this is required by this metadata. This metadata can optionally include instructions about the actual remote data binding and transport to be used, as well as requirements around security, reliability, transactions and other Qualities of Service, depending on metadata type. Intents are in any case used to help consumers discover compatible services.

If the DSW detects distributed OSGi metadata it has to configure a proxy for the service, set the appropriate service properties (derived from the metadata), and optionally register it with the service registry for it to be detected by a Discovery service implementation.

On the consumer-side, the distribution software is responsible for creating proxies to remote services so that they can be invoked by the consumer, and supporting the filtering of services by the consumer to detect a remote service, if desired.

The distribution software is optionally responsible for interacting with the Discovery Service to publish and subsequently discover services that it has made available over the network.

Distribution software is an optional component in the OSGi framework that would typically be deployed as one or more OSGi bundles.

The following diagram illustrates a possible solution to the design using Apache Felix as the OSGi platform and Apache CXF as the distribution software.
As illustrated in Figure 8, distributed communications between OSGi platform instances can be achieved by configuring a distributed software system such as Apache CXF into both client and server sides. In this example Apache Felix is used as the OSGi Framework implementation, and CXF is loaded into the framework. On the application side, common interface bundles are used, while consumer bundles are deployed on the client side and service bundles are deployed on the server side. A service proxy on the client side performs the communication with the remote endpoint deployed on the server side, which is created by CXF when the service is published. The Discovery service can be used on the client side to discover the location and additional properties of the remote service. In this example, the discover service references a metadata file in a directory local to the client, but another Discovery service would access a remote discovery mechanism such as UDDI or LDAP.

**5.2.2 Interface description**

The requirements for the distribution software state that the mechanism of how it implements the remote capabilities should not be defined in this document. Consequently, there is no mandatory functional interface to be implemented by a distribution software solution.

On the other hand, the need for identification of the deployed distribution software, its capabilities and version was raised and agreed upon. Therefore, any distribution software SHOULD implement the interface in Section 5.2.2.1 to return information about itself that is useful for identification and also in logging statements.

**5.2.2.1 Distribution Software Interface**

The distribution software implementation should be mandated to register a service in the local OSGi Service Registry that implements a standardized interface, which allows for obtaining static information
about the vendor, version, etc. as well as dynamic information about the remote service proxies it has created, the protocols it supports, and possibly other runtime statistics, which can be of value for a management console.

/**
 * Every Distribution Provider registers exactly one Service in the
 * ServiceRegistry implementing this interface. The service is registered with
 * extra properties identified at the beginning of this interface to denote the
 * Distribution Provider product name, version, vendor and supported intents.
 */

public interface DistributionProvider {

/**
 * Service Registration property for the name of the Distribution Provider
 * product.
 */
  static final String PROP_KEY_PRODUCT_NAME =
    "osgi.remote.distribution.product";

/**
 * Service Registration property for the version of the Distribution
 * Provider product.
 */
  static final String PROP_KEY_PRODUCT_VERSION =
    "osgi.remote.distribution.product.version";

/**
 * Service Registration property for the Distribution Provider product
 * vendor name.
 */
  static final String PROP_KEY_VENDOR_NAME =
    "osgi.remote.distribution.vendor";

/**
 * Service Registration property that lists the intents supported by this
 * DistributionProvider. Value of this property is of type
 * Collection (<? extends String>).
 */
  static final String PROP_KEY_SUPPORTED_INTENTS =
    "osgi.remote.distribition.supported_intents";

/**
 * @return ServiceReferences of services registered in the local Service
 * Registry that are proxies to remote services. If no proxies are
 * registered, then an empty collection is returned.
 */
  Collection /*<? extends ServiceReference>*/ getRemoteServices();

/**
 * @return ServiceReferences of local services that are exposed remotely
 * using this DistributionProvider. Note that certain services may be
 * exposed and without being published to a discovery service. This
 * API returns all the exposed services. If no services are exposed an
 * empty collection is returned.
 */
  Collection /*<? extends ServiceReference>*/ getExposedServices();

/**
 * Provides access to extra properties set by the DistributionProvider on
 * endpoints, as they will appear on client side proxies given an exposed
 * ServiceReference.
 * These properties are not always available on the server-side
 * ServiceReference of the exposed
 * service but will be on the remote client side proxy to this service.
* This API provides access to these extra properties from the exposing side.
* E.g. a service is exposed remotely, the distribution software is configured to add transactionality to the remote service. Because of this, on the client-side proxy the property service.intents="transactionality" is set. However, these intents are *not* always set on the original ServiceRegistration on the server-side since on the server side the service object is a local pojo which doesn’t provide transactionality by itself. This QoS is added by the distribution.
* This API provides access to these extra properties from the server-side.
* @param sr A ServiceReference of an exposed service.
* @return The map of extra properties.
*/
Map /*<String, String>*/ getExposedProperties(ServiceReference sr);

### 5.2.2.2 Exception Handling

There will be a new type of exception for the ServiceException: REMOTE. This type of exception is thrown when there is an issue with the distribution software used to covert between the protocol-specific and OSGi invocations.

When using a specific type of distribution software, the exception handling system must allow distribution software specific exceptions to be captured and propagated to the client as if OSGi was not involved. For example, RMI exceptions can still be reported.

However since distributed OSGi is adding a mapping layer between a service and the distribution software, it’s possible for an exception to occur within the mapping layer. The REMOTE exception is thrown to indicate a problem in this area, not to indicate problem within the distribution software itself.

### 5.3 Discovery Service

The Discovery service is an optional service, which enables publication of services running in a framework to remote consumers and discovery of services running outside a framework.

Publication of a service consists of publication of all the service metadata passed usually by the Distribution Software to Discovery. Discovery may use any internal protocol to transmit that service metadata. On the consumers side distribution software uses this metadata for filtering of potential candidates and creation of service proxies.

#### 5.3.1 Functionality

There are two models for sharing information about distributed services in a system. Either remote services required by a Distribution Software on a node are known upfront and definitions of matching services are transmitted to it, or that Distribution Software autonomously looks for services available in the network. Distributed OSGi supports the latter ‘discovery’ model in the optional Discovery Service but allows the other model to be supported as an implementation option of Distribution Software.

The Discovery service allows the distribution software to actively discover services based on filter criteria. In addition, the discovery service may provide an asynchronous notification mechanism, which alerts interested clients about the availability of particular remote services.

The strategy and details of the discovery service is left to the implementers. For instance discovery can be performed eagerly (i.e. before anyone has asked for the service), or lazily (i.e. triggered as part of a request to use the service). The design is intended to be simple and flexible enough to allow for multiple different implementations to reside in the same OSGi service platform concurrently. Each Discovery service implementation is expected to provide one or multiple discovery protocols, which are either well known (e.g. SLP, UDDI) or proprietary. Proprietary protocol implementations allow for reuse of existing
mechanisms while open standard implementations allow for better integration with existing products in the enterprise market.

The distribution software on the service provider side passes the information about the service provider to Discovery services. For this purpose distribution software registers a `ServicePublication` object, whose registration makes all existing Discovery services publish the provided information and make the service discoverable by other OSGi service platforms as well as other external clients understanding the used protocol. A service may be published by a distribution software multiple times with different bindings. In the case of multiple distribution software implementations in the same platform, multiple publications of a particular service may occur with the same as well as with different bindings.

Figure 9 illustrates also how multiple Discovery services are used to publish a service via more than one type of discovery mechanism (such as SLP and UDDI).

![Figure 9 OSGi service published over multiple protocols](image)

By implementing a discovery protocol of any open standard, the discovery is not bound to OSGi services alone. This allows discovery of services implemented and offered in different technologies like .NET or C++. Likewise, OSGi services are published using the standard protocol to clients built on other technologies than OSGi.

### 5.3.2 Interface description

#### 5.3.2.1 Discovery interface

```java
/**
 * Discovery registers a service implementing this interface. This service is
 * registered with extra properties identified at the beginning of this
 * interface to denote the name of the product providing Discovery
 * functionality, its version, vendor, used protocols etc..
 *
 * Discovery allows to publish services exposed for remote access as well as to
 * search for remote services. Register a (@link ServicePublication) service in
 * order to publish service metadata and or a (@link DiscoveredServiceTracker)
 * service in order to search for remote services.<BR>
 * Discovery service implementations usually rely on some discovery protocols or
 * other information distribution means.
 *
 * @version $Revision: 6046 $
 */

public interface Discovery {
```
/**
 * Service Registration property for the name of the Discovery product.
 */
static final String PROP_KEY_PRODUCT_NAME = "osgi.remote.discovery.product";

/**
 * Service Registration property for the version of the Discovery product.
 */
static final String PROP_KEY_PRODUCT_VERSION = "osgi.remote.discovery.product.version";

/**
 * Service Registration property for the Discovery product vendor name.
 */
static final String PROP_KEY_VENDOR_NAME = "osgi.remote.discovery.vendor";

/**
 * Service Registration property that lists the discovery protocols used
 * by
 * this Discovery service. Value of this property is of type Collection
 * (<?
 * extends String>).
 */
static final String PROP_KEY_SUPPORTED_PROTOCOLS = "osgi.remote.discovery.supported_protocols";
}

5.3.2.2 ServicePublication Interface

/**
 * Register a service implementing the <code>ServicePublication</code> interface
 * in order to publish metadata of a particular service (endpoint) via
 * Discovery. Metadata which has to be published is given in form of properties
 * at registration. <br>
 * In order to update published service metadata, update the properties
 * registered with the <code>ServicePublication</code> service. Depending on
 * Discovery's implementation and underlying protocol it may result in an update
 * or new re-publication of the service. <br>
 * In order to unpublish the previously published service metadata, unregister
 * the <code>ServicePublication</code> service. <br>
 * Please note that providing the {@link #PROP_KEY_SERVICE_INTERFACE_NAME} property is mandatory
 * when a <code>ServicePublication</code> service is
 * registered. Note also that a Discovery implementation may require provision
 * of additional properties, e.g. some of the standard properties defined below,
 * or may make special use of them in case they are provided. For example an
 * SLP-based Discovery might use the value provided with the
 * {@link #PROP_KEY_ENDPOINT_LOCATION} property for construction of a SLP-URL
 * used to publish the service. <br>
 * Also important is that it's not guaranteed that after registering a
 * <code>ServicePublication</code> object its service metadata is actually
 * published. Beside the fact that at least one Discovery service has to be
 * present, the provided properties have to be valid, e.g. shouldn't contain
 * case variants of the same key name, and the actual publication via Discovery
 * mechanisms has to succeed.
 * @version $Revision: 6485 $
 */
public interface ServicePublication {
/**
 * Mandatory ServiceRegistration property which contains a collection of
 * full qualified interface names offered by the advertised service
 * endpoint. Value of this property is of type Collection (<? extends
 * String>).
 */
public static final String PROP_KEY_SERVICE_INTERFACE_NAME
= "service.interface";

/**
 * Optional ServiceRegistration property which contains a collection of
 * interface names with their associated version attributes separated by
 * [separator] e.g. 'my.company.foo|1.3.5 my.company.zoo|2.3.5'. In
 * case no version has been provided for an interface, Discovery may use
 * the
 * String-value of <code>org.osgi.framework.Version.emptyVersion</code>
 * constant. <br>
 * Value of this property is of type Collection (<? extends String>).
 */
public static final String PROP_KEY_SERVICE_INTERFACE_VERSION
= "service.interface.version";

/**
 * Optional ServiceRegistration property which contains a collection of
 * interface names with their associated (non-Java) endpoint interface
 * names separated by [separator] e.g.:<br>
 * 'my.company.foo|MyWebService my.company.zoo|MyWebService'.<br>
 * This (non-Java) endpoint interface name is usually a communication
 * protocol specific interface, for instance a web service interface name.
 * Though this information is usually contained in accompanying properties
 * e.g. a wsdl file, Discovery usually doesn't read and interprets such
 * service meta-data. Providing this information explicitly, might allow
 * external non-Java applications find services based on this endpoint
 * interface.<br>
 * Value of this property is of type Collection (<? extends String>).
 */
public static final String PROP_KEY_ENDPOINT_INTERFACE_NAME
= "osgi.remote.endpoint.interface";

/**
 * Optional property of the published service. <br>
 * Property keys are handled in a case insensitive manner (as OSGi
 * Framework does). <br>
 * Value of this property is of type <code>java.util.Map</code>.<br>
 */
public static final String PROP_KEY_SERVICE.getProperties
= "service.properties";

/**
 * Optional property of the published service identifying its location.<br>
 * Value of this property is of type <code>java.net.URL</code>.<br>
 */
public static final String PROP_KEY_ENDPOINT_LOCATION
= "osgi.remote.endpoint.location";

/**
 * Optional property of the published service uniquely identifying its
* endpoint. Value of this property is of type <code>String</code>.
*/

```java
public static final String PROP_KEY_ENDPOINT_ID = "osgi.remote.endpoint.id";
```

/**
 * Separator constant for association of interface-specific values with
 * the
 * particular interface name. See also
 * {@link #PROP_KEY_SERVICE_INTERFACE_VERSION} and
 * {@link #PROP_KEY_ENDPOINT_INTERFACE_NAME} properties which describe
 * such
 * interface-specific values.
 */

```java
public static final String SEPARATOR = "|";
```

5.3.2.3 DiscoveredServiceTracker interface

/**
 * Interface of trackers for discovered remote services. <br>
 * When such a service is registered with the framework, then {@link Discovery} will notify it about remote services matching one of the provided criteria and will keep notifying it on changes of information known to Discovery regarding this services.<br>
 * <code>Discovery</code> may deliver notifications on discovered services to a <code>DiscoveredServiceTracker</code> out of order and may concurrently call and/or reenter a <code>DiscoveredServiceTracker</code>.<br>
 * @version $Revision: 6510 $
 */

```java
public interface DiscoveredServiceTracker {

    /**
     * Optional ServiceRegistration property which contains service interfaces this tracker is interested in. Value of this property is of type Collection (<? extends String>). <br>
     * Property is optional, may be null.
     */

    public static final String PROP_KEY_MATCH_CRITERIA_INTERFACES = "osgi.discovery.interest.interfaces";

    /**
     * Optional ServiceRegistration property which contains filters for services this tracker is interested in. <br>
     * Note that these filters need to take into account service publication properties which are not necessarily the same as properties under which a service is registered. See {@link ServicePublication} for some standard properties used to publish service metadata. <br>
     * The following sample filter will make Discovery notify the DiscoveredServiceTracker about services providing interface 'my.company.foo' of version '1.0.1.3':<br>
     * "(&{service.interface=my.company.foo}{service.interface.version=my.company.foo|1.0.1.3})". <br>
     * Value of this property is of type Collection (<? extends String>). <br>
     * Property is optional, may be null.
     */
```
public static final String PROP_KEY_MATCH_CRITERIA_FILTERS = "osgi.discovery.interest.filters";

/**
 * Receives notification that information known to Discovery regarding a
 * remote service has changed. <br>
 * The tracker is only notified about remote services which fulfill the
 * matching criteria, either one of the interfaces or one of the filters,
 * provided as properties of this service. <br>
 * If multiple criteria match, then the tracker is notified about each of
 * them. This can be done either by a single notification callback or by
 * multiple subsequent ones. <br>
 * @param notification the <code>DiscoveredServiceNotification</code>
 * object describing the change.
 */
void serviceChanged(DiscoveredServiceNotification notification);
}

5.3.2.4 DiscoveredServiceNotification interface

/**
 * Interface for notification on discovered services.
 */
@version $Revision: 6510 $

public interface DiscoveredServiceNotification {

/**
 * Notification indicating that a service matching the listening criteria
 * has been discovered.
 */
public final static int AVAILABLE = 0x00000001;

/**
 * Notification indicating that the properties of a previously discovered
 * service have changed.
 */
public final static int MODIFIED = 0x00000002;

/**
 * Notification indicating that a previously discovered service is no
 * longer known to discovery.
 */
public final static int UNAVAILABLE = 0x00000004;

/**
 * Notification indicating that the properties of a previously discovered
 * service have changed and the new properties no longer match the listener's
 * filter.
 */
public final static int MODIFIED_ENDMATCH = 0x00000008;

/**
 * Returns information currently known to Discovery regarding the service endpoint.
 * <p>
 * @return metadata of the service this Discovery notifies about.
 */
ServiceEndpointDescription getServiceEndpointDescription();

/**
 * Returns the type of notification. The type values are:
 * <ul>
 * <li>{@link #AVAILABLE}</li>
 * <li>{@link #MODIFIED}</li>
 * <li>{@link #MODIFIED_ENDMATCH}</li>
 * <li>{@link #UNAVAILABLE}</li>
 * </ul>
 * <p>
 * @return Type of notification regarding known service metadata.
 */
int getType();

/**
 * Returns interface name criteria of the {link DiscoveredServiceTracker} object matching with the interfaces of the ServiceEndpointDescription and thus caused the notification.
 * <p>
 * @return matching interface name criteria of the {link DiscoveredServiceTracker} object being notified.
 */
Collection /* <String> */getInterfaces();

/**
 * Returns filters of the {link DiscoveredServiceTracker} object matching with the ServiceEndpointDescription and thus caused the notification.
 * <p>
 * @return matching filters of the {link DiscoveredServiceTracker} object being notified.
 */
Collection /* <String> */getFilters();

5.3.2.5 ServiceEndpointDescription interface

/**
 * This interface describes an endpoint of a service. This class can be considered as a wrapper around the property map of a published service and its endpoint. It provides an API to conveniently access the most important properties of the service.
 * <p>
 * @version $Revision: 6037 $
 */
public interface ServiceEndpointDescription {

/**
 * Returns the value of the property with key {link ServicePublication#PROP_KEY_SERVICE_INTERFACE_NAME}.
 */
* @return service interface names provided by the advertised service (endpoint). The collection is never null or empty but contains at least one service interface. */

Collection /* <? extends String> */ getProvidedInterfaces();

/**
 * Returns non-Java endpoint interface name associated with the given interface. Value of the property with key
 * {link ServicePublication#PROP_KEY_ENDPOINT_INTERFACE_NAME} is used by this operation.
 *
 * @param interfaceName for which its non-Java endpoint interface name should be returned.
 *
 * @return non-Java endpoint interface name. Null, if it hasn't been provided.
 */

String getEndpointInterfaceName(String interfaceName);

/**
 * Returns version of the given interface. Value of the property with key
 * {link ServicePublication#PROP_KEY_SERVICE_INTERFACE_VERSION} is used by this operation.
 *
 * @param interfaceName for which its version should be returned.
 *
 * @return Version of given service interface. Null, if it hasn't been provided.
 */

String getVersion(String interfaceName);

/**
 * Returns the value of the property with key
 * {link ServicePublication#PROP_KEY_ENDPOINT_LOCATION}.
 *
 * @return The URL of the service location. Null, if it hasn't been provided.
 */

URL getLocation();

/**
 * Returns the value of the property with key
 * {link ServicePublication#PROP_KEY_ENDPOINT_ID}.
 *
 * @return Unique id of service endpoint. Null, if it hasn't been provided.
 */

String getEndpointID();

/**
 * Getter method for the property value of a given key.
 *
 * @param key Name of the property
 *
 * @return The property value, null if none is found for the given key
 */

Object getProperty(String key);
/**
 * @return java.util.Collection of property names available in the ServiceEndpointDescription. The collection is never null or empty but contains at least basic properties like objectClass for the service interface. The collection represents a snapshot and as such is not going to be updated in case properties were added or removed at a later point of time.
 */
Collection/* <? extends String> */getPropertyKeys();

/**
 * @return Returns all properties of the service as a java.util.Map. The map is never null or empty but contains at least basic properties like objectClass for the service interface. The collection represents a snapshot and as such is not going to be updated in case properties were added or removed at a later point of time.
 */
Map/* <String, Object> */ getProperties();

5.3.3 Discovery using a local file(s)
The requirement for providing discovery type information in local files comes from the 'I want to connect to a Google Service' use-case. Basically, a mechanism is needed to specify the details of a service without access to an external discovery service.

To satisfy this requirement, an optional mechanism can be provided that uses resources inside a bundle providing Discovery type information. This mechanism is often provided by the Distribution Software, but can also be provided by another entity.

The mechanism checks bundles for the existence of xml files in the OSGI-INF/remote-service directory and if found it provides the information in these files to the Distribution Software, similar to how Discovery provides this information to the DSW.

The local files can be used to statically configure a client in case of not using the discovery model to share service information and distribute this information in a controlled way based on upfront knowledge of services required by a node.

The following is an example of such a file which shows a single Distribution Software approach used by the initial Reference Implementation. Section (add ref to section called "Bindings in Service Descriptions") shows an example of how a Distribution Software can also optionally choose to use SCA metadata.

<?xml version="1.0" encoding="UTF-8"?>
<service-descriptions xmlns="http://www.osgi.org/xmlns/ad/v1.0.0">
    <service-description>
        <provide interface="com.iona.soa.pojo.hello.HelloService"/>
            <property name="service.intents">SOAP HTTP</property>
            <property name="osgi.remote.configuration.type">pojo</property>
            <property name="osgi.remote.configuration.pojo.address">http://localhost:9000/hello</property>
    </service-description>
</service-descriptions>
The XML files use the http://www.osgi.org/xmlns/sd/v1.0.0 namespace which can be found in section 5.8.

The solution should use a folder named "OSGI-INF/remote-service" and parse all files with the *.xml extension in this folder per default (i.e. adopt the Extender model).

The location for the service description folder and individual files within it can be overridden by a specific Manifest header named "Remote-Service". Multiple clauses in this header are allowed (comma separated). The format of the header should follow the Bundle.findEntries() approach. So the default value of this property would be /OSGI-INF/remote-service/*.xml. An example possible user-provided value for this property could be:

/META-INF/osgi/services.remote,/MyDirectory/osgi/*.xml

This matches a single file called services.remote in the /META-INF/osgi directory plus all files ending with *.xml in the /MyDirectory/osgi folder.

5.4 Service Registry Hooks

5.4.1 Registration of Remote Services in Local Service Registry

In the OSGi specification R4.1 the Service Registry serves as a central entity where one could register (locally available) services as well as search for them. Reusing the same mechanism for remote services would help to stay as much as possible in the established OSGi programming model and hence help developers in adopting the new capabilities coming with RFC 119. Using the Service Registry for both, local and remote, services offers also a certain degree of transparency for service providers and consumers.

The implementation of RFC 119 uses the ListenerHook as defined in RFC 126. This allows the distribution software to be informed when a consumer is looking for a service that potentially is not available in the local container (yet) and may therefore be discovered in the network.

5.4.2 Additional filtering

If additional filtering to what service consumers specify in their LDAP filter is required, this can be achieved by installing a FindHook as described in RFC 126. A FindHook allows the implementor to restrict the visible set of services for one or more bundles. A possible use for the FindHook is to prevent a particular bundle from seeing remote services if use of remote services is not desired for this bundle.

For further details regarding the specification of this Service Registry Hook see RFC 126.

5.5 Service Programming Model

Sharing of a common service contract between service consumers and providers is fundamental for their interaction. Typically a service contract consists of two parts:

- Description of the functionality the service provider offers. That's mostly expressed by a service interface description e.g. a Java interface and the service’s documentation.
• Description of the non-functional or quality of service (QoS) requirements regarding the way the agreed functionality is provided e.g. data has to be encrypted, call semantics.

An important point for RFC119 is its explicit support for dynamic wiring. In contrast to static wiring, where the concrete communication partners as well as their service contract are known beforehand (at the latest at deployment time) dynamic wiring allows service consumers and providers to establish contracts at runtime based on some criteria e.g. interface, supported communication protocols, or a set of QoS requirements (typically expressed using intents). An actual service contract results from requirements of a service provider and consumer as well as from the capabilities of distribution software on both sides.

The following kinds of metadata have been defined for service contracts:

• Service interface – describes the functionality of a service.
• Properties – provide information about the service object.
• Intents – state abstract requirements on service provider and consumer capabilities.

The above metadata may be sufficient when using the same distribution software on both client and service provider. To facilitate portability of configuration and interoperability in the case where multiple Distribution Software implementations are deployed, a service can be optionally configured using additional SCA metadata (see section Error! Reference source not found. Error! Reference source not found.). Other metadata types are also permitted (standard or proprietary), through an extensibility mechanism but their integration into OSGi is not defined.

5.5.1 Service interface description

The service interface description defines the functionality, which a service provides. A service interface is the most basic service metadata and has to be well known by both interaction partners.

For RFC 119 a service interface is defined using a Java interface. The Java interface is typically used to derive a Distribution Provider interface, and some restrictions on the Java interface are therefore necessary to ensure compatibility across multiple DSW types (see Section 5.8).

5.5.2 Properties

Property – properties are used to describe a service while registering it in OSGi service registry. For more details on service properties, please refer to OSGi 4.1 core specification chapter 5.2.5.

Service properties can be provided statically by the bundle implementation and/or dynamically as configuration data that is used during the service registration, for example using the Configuration Admin service of OSGi R4.

Note that the properties defined in this section are for use with remote services only.

5.5.2.1 Definition of new Properties

Any custom service property can easily be defined. Please refer for more details to OSGi 4.1 core specification chapter 5.2.5. These have no bearing on the distribution of a service.

5.5.2.2 Standard Properties

• service.intents – an optional list of intents provided by the service. The property advertises capabilities of the service and can be used by the service consumer in the lookup filter to only select a service that provides certain qualities of service. The value of this property is of type String[] and has to be provided by the service as part of the registration, regardless whether it’s a local service or a proxy. The value on the proxy is a union of the value specified by the service provider, plus any remote-specific intents (see orgi.remote.require.intents, below), plus any intents which the Distribution Software adds that describe characteristics of the Distribution being mechanism. Therefore the value of this property can vary between the client side proxy and the server side.
• **osgi.remote.interfaces** – [ "*" | interface_name [, interface_name]* ]: A distribution software implementation may expose a service for remote access, if and only if the service has indicated its intention as well as support for remote invocations by setting this service property in its service registration. The value of this property is of type String[]. If the list contains only one value, which is set to "*", all of the interfaces specified in the BundleContext.registerService() call are being exposed remotely. The value can also be set to a comma-separated list of interface names, which should be a subset of the interfaces specified in the registerService call. In this case only the specified interfaces are exposed remotely.

• **osgi.remote.requires.intents** – an optional list of intents that should be provided when remotely exposing the service. If a DSW implementation cannot satisfy these intents when exposing the service remotely, it should not expose the service. The value of this property is of type String[].

• **osgi.remote** – this property is set on client side service proxies registered in the OSGi Service Registry.

• **osgi.remote.configuration.type** – service providing side property that identifies the metadata type of additional metadata, if any, that was provided with the service, e.g. "sca". Multiple types and thus sets of additional metadata may be provided. The value of this property is of type String[].

Both the **osgi.remote.interfaces** and **osgi.remote.requires.intents** should be modifiable by the deployer after the service has been developed. This can either be done through the Configuration Admin Service or through another mechanism.

Only the **osgi.remote.interfaces** property is required.

The **service.intents** property optionally defines the QoS capabilities that a published service provides, and allows a service requester to filter services according to its desired QoS capabilities.

The **osgi.remote.configuration.type** optionally defines portable metadata to address the requirement for consistency across multiple DSW types. Distributed OSGi specifies how to use SCA metadata for this purpose.

The following example illustrates a potential XML file that could be used by Declarative Services to register the properties for distributed OSGi capability. This file would be installed through a bundle and identified by the bundle’s Service-Component manifest header:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<component name="OrderBeerService">
  <implementation .../>
  <service>
    <provide interface="org.beer.OrderBeerService" >
      <property name="osgi.remote.interfaces">*</property>
      <property name="osgi.remote.requires.intents">confidentiality</property>
    </provide>
  </service>
</component>
```

The example illustrates the **osgi.remote.interfaces** and **osgi.remote.requires.intents** properties specified for the BeerOrderService, which are associated with its interface. The confidentiality intent specifies the capability of the service to support encryption, such as through HTTP or IIOP/SSL.
5.5.3 Intents

An intent is an abstraction of a distributed computing capability that can be used to provision and select services. It describes one or more requirements of a service provider or consumer on the distribution software serving them.

An intent is a high-level, generic statement of ‘what’ a consumer may require from a provider. An intent is also a statement of what can be required of a deployer by a developer. An intent is associated with a service during deployment and can be used as a filter by a service consumer during the service discovery operation. When using an intent to filter a service, the consumer expects that the DSW has implemented the specified intent. The definition of intents comes from SCA but OSGi developers can also define their own intents, and any intent can be mapped by a given DSW to a DSW specific mechanism to fulfill the intent. Examples include intents for a reliable communication protocol, secure transmission, or a specific binding type.

The intent syntax is defined by the Service Component Architecture (SCA) and is extensible. This RFC references the SCA intents, defines two intents, and describes how to define additional intents.

A service requester can use an intent to help selecting a compatible service provider, and a service provider can use an intent to provision and deploy a service that advertises the intent.

When the same type of distributed software system is configured for both requester and provider, the DSW is not required to use the SCA mechanisms for defining the concrete instantiation of the intents, as long as its abstract meaning can be fulfilled by the DSW using another, similar mechanism. For example, instead of using WS-Policy as SCA does, a CORBA DSW might use CORBA policies. Any distribution detail undefined through intents or additional metadata is left to Distribution Software’s interpretation and its (default) configuration. For example, a provided service may state that it requires the service to be 'confidential'. The Distribution Software may choose to distribute the service using a SOAP-based protocol with encryption. In doing so, it may optionally add the 'soap' intent so that clients selecting based on the 'soap' intent will also match this service.

The advantage of this Intents-based approach is that designers and developers can easily state requirements on service exposure or service reference (proxy) without the need to understand the complexities of mechanisms actually provided by the distribution software.

Intents may be provided by the component designer or by the deployer through configuration, i.e. Configuration Admin service, and used by the requester to select a service. For example, if a service requester requires 'integrity', only services which have been given the integrity intent (and provisioned accordingly) will be returned by the distribution software.

Intents that are provided by a service are listed in a service property named service.intents.

5.5.3.1 Example of using Intents

The example below shows a Declarative Service component called BeerOrderService which exposes a service with an org.beer.BeerOrderService interface, and consumes a BeerWarehouseService with an org.beer.BeerWarehouseService interface.

The service property osgi.remote.requires.intents is set to specify that remote communication with the BeerOrderService component should provide 'confidentiality' and each delivery should occur 'exactlyOnce' (no duplicate messages or dropped messages). This might be desirable in order to prevent people snooping on messages and in order to ensure that orders are not lost or duplicated. These intents would typically be implemented using encryption and a reliable transport, respectively.

The example also shows how the component uses a BeerWarehouseService which is required to be available over a transport which assures messages are delivered 'exactlyOnce'. This is expressed using the target filter "(service.intents=exactlyOnce)". Only a service which provides this intent will be injected for the BeerWarehouseService reference.

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
<component name="BeerOrderService">
    <implementation .../>
    <service>
        <provide interface="org.beer.BeerOrderService"/>
        <property name="osgi.remote.interfaces">*</property>
        <property name="osgi.remote.requires.intents">
            confidentiality
            exactlyOnce
        </property>
        <provide />
    </service>
    <reference .../>
    <reference name="BeerWarehouseService" interface="org.beer.BeerWarehouseService" target="(service.intents=exactlyOnce)" />
</component>

5.5.3.2 Defining Intents

An intent is a string with an associated abstract meaning. Their definition can be as simple as choosing a string name and documenting its meaning so that it can be shared between the various roles involved in creating the distributed system. Any user of Distributed OSGi is free to define their own intents using the mechanism defined in Section 5.7.6.

5.5.3.3 OSGi-defined Intents

OSGi defines two intents, passByReference and passByValue to allow services and clients to specify which type of call semantics they require.

- passByReference - states that the service requires pass-by-reference semantics. This restricts the subset of usable bindings to those that support pass-by-reference semantics, such as RMI.
- passByValue – states that the service requires pass-by-value semantics. This restricts the subset of usable bindings to those that support pass-by-value semantics.

When neither of these intents is used, then passByValue semantics are assumed, and a Distribution Software which publishes the service must ensure it adds this intent to the ServiceEndpointDescription to allow clients to explicitly select on it.

5.5.3.4 SCA-defined Intents

SCA defines a set of intents (strings and their associated abstract meaning). OSGi re-uses these intent definitions where appropriate (e.g. for defining service contracts QoS such as ‘confidentiality’, or specific protocol requirements such as ‘soap.1_1’).

SCA also defines a schema for adding new intents, the use of which is described in the section on Service Distribution using SCA Metadata (see section 5.7). A Distribution Software may choose to use this schema as a mechanism for adding new intent definitions. The types of intent relationships that schema allows and how they are exploited in Distributed OSGi are described below.

Below are the set of intents defined by the SCA Policy Framework specification Working Draft 09. Note, the ‘.’ is used to define ‘qualified intents’ which are described in more detail in section 5.5.3.10. The expectation is this list will be updated to the most current document agreed by the SCA Policy TC (either a Community Draft or Public Review Draft) for inclusion in the final OSGi compendium specification.
5.5.3.5 Security Intents

**authentication**: the authentication intent is used to indicate that a client must authenticate itself in order to use a service. Typically, the client security infrastructure is responsible for the server authentication in order to guard against a "man in the middle" attack.

**authentication.message**: indicates that authentication should be realized at the message level of the communication.

**authentication.transport**: indicates that authentication should be realized at the transport layer of the communication.

**confidentiality**: the confidentiality intent is used to indicate that the contents of a message are accessible only to those authorized to have access (typically the service client and the service provider). A common approach is to encrypt the message, although other methods are possible.

**confidentiality.message**: indicates that confidentiality should be realized at the message level of the communication.

**confidentiality.transport**: indicates that confidentiality should be realized at the transport layer of the communication.

**integrity**: the integrity intent is used to indicate that assurance is required that the contents of a message have not been tampered with and altered between sender and receiver. A common approach is to digitally sign the message, although other methods are possible.

**integrity.message**: indicates that integrity should be realized at the message level of the communication.

**integrity.transport**: indicates that integrity should be realized at the transport layer of the communication.

5.5.3.6 Reliable Messaging Intents

**atLeastOnce**: the atLeastOnce intent is used to assure that a message that is successfully sent by a service consumer is delivered to the destination (i.e. service implementation). The message could be delivered more than once to the service implementation. A message that is successfully sent by a service implementation is also assured to be delivered to the destination (i.e. service consumer). The message could be delivered more than once to the service consumer.

**atMostOnce**: the atMostOnce intent is used to assure that a message that is successfully sent by a service consumer is not delivered more than once to the service implementation. However, the message is not guaranteed to be delivered to the service implementation. A message that is successfully sent by a service implementation is also assured not to be delivered more than once to the service consumer. The binding implementation does not guarantee that the message is delivered to the service consumer.

**exactlyOnce**: the exactlyOnce intent assures that a message sent by a service consumer is delivered to the service implementation. It also assures that the message is not delivered more than once to the service implementation. A message sent by a service implementation is also assured delivered to the service consumer. It also assures that the message is not delivered more than once to the service consumer.

ExactlyOnce is a profile intent of atLeastOnce and atMostOnce, the combination of which results in the exactlyOnce semantic.

**ordered**: the ordered intent assures that messages are delivered to the service implementation in the order in which they were sent by the service consumer. This intent does not guarantee that messages that are sent by a service consumer are delivered to the service implementation. Messages are also assured to be delivered to the service consumer in the order in which they were sent by the service implementation. This intent does not guarantee that messages that are sent by the service implementation are delivered to the service consumer.
5.5.3.7 Transactional Intents

**propagatesTransaction**: the propagatesTransaction intent indicates that the OSGi runtime must ensure that if the client of a service is running under a transaction, then that transaction context must be transmitted along with the invocation of an operation of the service and that the service invocation is dispatched under any propagated client transaction. Use of the propagatesTransaction intent implies that the Distribution Software must be capable of sending and receiving a transaction context and that a service with this intent specified will always join a propagated transaction, if present.

**suspendsTransaction**: the suspendsTransaction intent indicates that the OSGi runtime must ensure the service client must not send any transaction context with any service invocation and that the service is not dispatched under any propagated client transaction.

5.5.3.8 Miscellaneous Intents

**SOAP**: the SOAP intent specifies that the SOAP messaging model should be used for delivering messages. It does not require the use of any specific transport technology for delivering the messages, so for example, this intent can be supported by a binding that sends SOAP messages over HTTP, over bare TCP or over JMS. If the intent is required is an unqualified form of the intent, then any version of SOAP is acceptable.

**SOAP.1_1**: the SOAP.1_1 intent specifies the use of SOAP version 1.1 only. SOAP versions are mutually exclusive (e.g. it is an error to specify both SOAP.1_1 and SOAP.1_2).

**SOAP.1_2**: the SOAP.1_2 intent specifies the use of SOAP version 1.2 only. SOAP versions are mutually exclusive (e.g. it is an error to specify both SOAP.1_1 and SOAP.1_2).

**JMS**: The JMS intent does not specify a wire-level transport protocol, but instead requires that whatever binding technology is used, the messages should be able to be delivered and received via the JMS API.

**NoListener**: the NoListener intent is only applicable to a service reference. It indicates that the client is not able to handle new inbound connections. It requires that the Distribution Software be configured so that any response (or callback) comes either through a back channel of the connection from the client to the server or by having the client poll the server for messages.

Note, this intent does not follow the case convention of starting with lower-case and is expected to change to “noListener”.

5.5.3.9 Not Applicable Intents

In addition to the intents described above, SCA defines intents which cover advanced interaction patterns, such as Conversations (stateful protocol-style interactions) as well as service implementation requirements on a transactional runtime. These intents are not applicable to Distributed OSGi, but are listed for completeness. An OSGi-based runtime which supports these capabilities should consider reusing these intents where applicable:

**Conversational**: the conversation intent is used to indicate that a service is conversational. A conversational service is one which is stateful and may have a defined ordering in which service methods should be called.

Note, this intent does not follow the case convention of starting with lower-case and is expected to change to “conversational”. This has been raised with the Policy TC co-chair.

**managedTransaction**: the managedTransaction intent states that the service implementation requires a managed transaction environment in order to run. The specific type of managedTransaction required is not constrained. The valid qualifiers for this intent are mutually exclusive

**managedTransaction.local**: the managedTransaction.local intent states that the service implementation cannot tolerate running as part of a global transaction, and will therefore run within a local transaction containment (LTC) that is started and ended by the OSGi runtime. Any global transaction context that is propagated to the hosting OSGi runtime must not be visible to the target service implementation. Any
interaction under this policy with a resource manager is performed in an extended resource manager local transaction (RMLT). Upon successful completion of the invoked service method, any RMLTs are implicitly requested to commit by the OSGi runtime. Note that, unlike the resources in a global transaction, RMLTs so coordinated in a LTC may fail independently. If the invoked service method completes with a non-business exception then any RMLTs are implicitly rolled back by the OSGi runtime. In this context a business exception is any exception that is declared on the service interface and is therefore anticipated by the service implementation. Local transactions cannot be propagated outbound across remote service invocations.

**managedTransaction.global**: the managedTransaction.global intent states that the service implementation requires an atomic transaction in order to run. The OSGi runtime must ensure that a global transaction is present before dispatching any method on the service. The OSGi runtime uses any transaction propagated from the client or else begins and completes a new transaction. See the **propagatesTransaction** intent below for more details.

**noManagedTransaction**: the noManagedTransaction intent states that the service implementation runs without a managed transaction, under neither a global transaction nor an LTC. A transaction that is propagated to the hosting OSGi runtime must not be joined by the hosting runtime on behalf of this service implementation. When interacting with a resource manager under this policy, the application (and not the OSGi runtime) is responsible for controlling any resource manager local transaction boundaries, using resource-provider specific interfaces (for example a Java implementation accessing a JDBC provider must choose whether a Connection should be set to autoCommit(true) or else must call the Connection commit or rollback methods).

**transactedOneWay**: the transactedOneWay intent indicates that OneWay invocations should be performed under transactional control. When applied to a service reference indicates that any OneWay invocation messages must be transacted as part of a client global transaction. If the client is not configured to run under a global transaction or if the Distribution Software does not support transactional message sending, then this is a deployment error. When transactedOneWay is applied to a service this indicates that any OneWay invocation message must be received from the transport binding in a transacted fashion, under the target service’s global transaction. The receipt of the message from the Distribution Software is not committed until the service transaction commits; if the service transaction is rolled back the the message remains available for receipt under a different service transaction. If the service is not configured to run under a global transaction or if the Distribution Software does not support transactional message receipt, then this is a deployment error.

**immediateOneWay**: the immediateOneWay intent indicates that OneWay invocations should be performed outside transactional control. When applied to a service reference indicates that any OneWay invocation messages is sent immediately regardless of any client transaction. When applied to a service indicates that any OneWay invocation is received immediately regardless of any target service transaction. The outcome of any transaction under which an immediateOneWay message is processed has no effect on the processing (sending or receipt) of that message.

### 5.5.3.10 Qualified Intents

An intent and the meaning it conveys can be specialized using a concept known as 'qualified intents'.

Example: Intent ‘confidentiality’ can be further qualified by extending it to 'confidentiality.message' and would mean that 'confidentiality' should be realized at the message level of the communication protocol e.g. by encrypting the messages. An alternative specialization of the intent 'confidentiality' might be 'confidentiality.transport' meaning that confidentiality should be realized through an encrypted transport.

Since qualification of intents is a specialization an intent 'confidentiality.message’ always fulfills the intent 'confidentiality' but not necessarily the other way round.

Qualification of intents is a recursive model so qualified intents may be qualified again e.g. 'confidentiality.message.body'.

---

**Propagates Transaction**: This intent indicates that any transaction propagated to the hosting OSGi runtime must be joined by the hosting runtime on behalf of this service implementation. When the application has completed any transactional work, the OSGi runtime should inform the hosting runtime of any changes that have occurred in the transaction. The hosting runtime must then either commit the transaction or roll it back.
5.5.3.11 Publishing of Qualified Intents

When publishing a service with qualified intents, the Distribution Software must make sure to list all appropriate intents for service selection. There are two aspects to this:

1. If a service has originally provided a qualified intent, then the Distribution Software should list also all the more general intents. A qualified intent is a specialization which means that a client looking for the more general intents should find a match. For example, 'confidentiality.message' would be published as 'confidentiality.confidentiality.message' so that a client which does not care how confidentiality is provided will match the service which specifically provides it through the messages.

2. If a service has originally provided a general intent and Distribution Software has implemented it according to a qualified version of that intent then it should list also all the applicable qualified intents in addition to the original general intent. For example service which initially stated 'confidentiality' should be published as 'confidentiality.confidentiality.message' if the Distribution Software implemented 'confidentiality' at the message level. So clients looking directly for qualified intents can also be served.

5.5.3.12 Profile Intents

A profile intent is an intent which is defined in terms of a set of intents. Using a profile intent is semantically equivalent to specifying all of the intents in the set. Profile intents are a convenience mechanism and remove the need to repeatedly specify the same intent sets in a solution.

The following example shows a new profile intent called 'communicationProtection' which combines the 'confidentiality' and 'integrity' intents. Its purpose is to ensure that communications cannot be viewed or tampered with:

```
<intent name="communicationProtection"
        constrains="binding"
        requires="confidentiality integrity">
    <description>
        Ensure that communications cannot be seen or tampered with by unauthorized personnel.
    </description>
</intent>
```

5.5.3.13 Publishing of Profile Intents

When publishing a service with a profile intent, the Distribution Software must make sure to list all appropriate intents for service selection. For example, the profile intent 'communicationProtection' would be published as 'communicationProtection confidentiality confidentiality.transport confidentiality.message integrity integrity.message integrity.transport'. Note, confidentiality and integrity are qualifiable intents and are therefore published according to the rules for qualified intents, described in Section 5.5.3.10.

If a Distribution Software natively understands intents then it must be configured with the profile intent definition. This enables the Distribution Software to know when all intents have been satisfied. If a Distribution Software does not natively understand intents, it is the responsibility of the deployer to ensure the Distribution Software is configured appropriately in order to satisfy the intents.

5.5.3.14 Exclusive Intents

Because intents are an expression of a service characteristic or requirement it is possible that two intents may be mutually exclusive and should not be used together. The existence of mutually exclusive intents on the same service is considered a deployment error. If a Distribution Software understands that two intents are mutually exclusive then it should not distribute the service.
5.5.4 Configuration type

The configuration type identifies the metadata used to describe additional DSW capabilities beyond intents, such as explicit communication protocol and data format bindings and quality of service policies. The main example in RFC 119 is SCA, but since RFC 119 is designed to support multiple DSW types, other metadata can be used and associated with additional configuration types.

The configuration type is specified using a property named `osgi.remote.configuration.type`. The property is set to string values which identify the type of additional metadata used. For example, the following shows how to specify that the additional metadata is in the form of SCA:

```
osgi.remote.configuration.type=sca
```

The configuration type is extensible so that other standard or proprietary types may be used. For example, the Foo Corporation may choose,

```
osgi.remote.configuration.type=foocorp
```

The configuration type property can be an array to allow multiple configuration types to be specified for the same service. For example, `osgi.remote.configuration.type=sca foocorp`

A naming convention is used to specify the service properties which provide the additional configuration. This follows the form `osgi.remote.configuration.<type string>.<sub-properties>`, so for `foocorp`, this would be,

```
osgi.remote.configuration.foocorp=...
```

It is recommended that each configuration type use sub-property names for their additional configuration as this is likely to make the properties more meaningful and allows multiple independent properties to be specified. For example, Foo Corp may require two items of additional configuration, which could be specified as,

```
osgi.remote.configuration.foocorp.config1=...
osgi.remote.configuration.foocorp.config2=...
```

A naming convention is used to specify the service properties which provide the additional configuration. This follows the form `osgi.remote.configuration.<type string>.<sub-properties>`, so for `foocorp`, this would be,

```
osgi.remote.configuration.foocorp=...
```

In general, metadata in a configuration type can be used to create a machine-readable description of a remoted service. This description can be compared with other descriptions for compatibility (i.e. is the service provider compatible with the service requester). That is, do they support the same communication protocols, encryption mechanisms, etc?

Service descriptions can be matched for compatibility initially at the intent level (i.e. intents can be compared for compatibility) but if the configuration type property is present and indicates additional metadata is available, it should be possible to perform an additional level of comparison for compatibility on the additional metadata.

If multiple matches are returned, matches based on intent properties are ranked lower than matches found using additional metadata. It should also be possible to rank services using a comparator.

5.5.5 Service Factories

In a non-distributed case Service Factories return a separate Service instance per consuming bundle.

For distributed OSGi Services, this behaviour is not extended into the remote case. For remoted services implemented using a ServiceFactory, the behaviour when a consumer calls the remote service by invoking on its remote endpoint should be similar to the Distribution Software bundle calling `Bundle.getBundleContext().getService()`. This means that a single instance of the remoted service will be used to serve remote invocations on it.

5.6 Collaboration of new and changed entities

In the following the interaction between service providers, consumers, distribution software and discovery is illustrated. Though involvement of discovery by distribution software is optional it's shown here for illustration reasons.
5.6.1 Interactions on the service provider side

5.6.1.1 Exposing a Service remotely

How the service B is registered in the OSGi framework and then made available for remote access is shown in Error! Reference source not found.. The important part in the picture is that service B is augmented with additional properties, which describe which interface of the service should be remotely exposed, required QoS, etc. (see section 5.5 for details about service metadata). This enables the distribution software to pick the appropriate protocol for service exposure and create an endpoint. In an optional step all service metadata required to communicate with that endpoint is published using the Discovery service. This happens by registering a ServicePublication service carrying service metadata to be published and which will be picked up by any existing Discovery service. The sequence diagram above shows the creation of only one endpoint by Distribution Provider, may be multiple, and publication by only one Discovery Service.

5.6.1.2 Modification of service properties

The following diagram illustrates how Distribution software may react to modification of properties of a service which was exposed for remote access and published via Discovery.

The diagram shows only one possible scenario. Depending on which properties have changed and Distribution Providers implementation, it may choose not to update the existing endpoint but to create and publish a new one. The previous endpoint would be unpublished and destroyed in that case. Discovery service may also choose to make a new publication instead of updating an existing one depending on what the discovery protocol supports.
5.6.1.3 Service Unregistration

The following figure depicts the flow of events in the case that a previously registered, remotely exposed and published service B is unregistered.

![Unregistration of a service](image)

**Figure 12: Unregistration of a service**
5.6.2 Interactions on the service consumer side

5.6.2.1 Lookup for a remote Service

In Figure 13: Client side service lookup is shown what happens on the service consumer side, if the consumer is hosted in an OSGi platform. The distribution software is using the optional Discovery service to locate an implementation of service B, which satisfies the requirements specified in the service lookup. If found, the distribution software creates a proxy for the available protocol (binding) that implements the service interface as well as the required Qualities of Service. This proxy is then registered in the local OSGi service registry and thus is accessible by the service consumer. Proxy’s properties reflect all concrete distribution-related service metadata like used binding, applied policies, service’s host etc.

On any subsequent lookup for the same interface this proxy may also be returned to other bundles provided that it’s capable to fulfill their QoS requirements as well.

Note: The proxy implementation is entirely left to the distribution software.

5.6.2.2 Service invocation

Service invocation is exactly the way it is today with local OSGi services. The exception to this is that an invocation that goes to a remote service can potentially throw a new RuntimeException: `osgi.framework.ServiceException` with as exception type `REMOTE` in the event there is a problem with the remote invocation. This exception can wrap any distribution technology-specific exception. A technology-specific exception may also be thrown directly if Distribution Provider is configured in this way.

As the new exception is a RuntimeException, existing code is not required to check for it, however, distribution-aware code has the option to catch it and react appropriately.
5.6.2.3 Service Unregistration

The following figure depicts the flow of events in the case that a previously discovered and bound service B becomes unavailable. The scenario assumes that the Discovery Service is informed about the fact that the remote service is no longer a valid reference.

![Unregistration of a service](image)

**Figure 14: Unregistration of a service**

It is the Distribution Software’s responsibility to ensure any proxies associated with unpublished services are unregistered.

Proxies to remote services which were configured from a locally installed bundle (e.g. through a local discovery service) are removed from the registry when the bundles which contributed them are stopped.

5.6.3 Interactions with Non-OSGi service providers and consumers

From interaction point of view there is no difference whether the other side uses OSGi or Non-OSGi technologies. Though it may put more restrictions on the exchanged data between service provider and consumer as well as Distribution Provider and Discovery e.g. no complex java objects.

5.6.4 Lifecycle dynamics

When a Distribution Provider is activated, it should:

- check for any services that are already registered in the OSGi Service Registry and have the property `osgi.remote.interfaces` specified and thus need to be exposed remotely.
- check for any active bundle which have unresolved references to remote services.

Before a Distribution Provider is deactivated, it should:

- unregister any proxies it has registered in the OSGi Registry (though Distribution Provider may rely on OSGi framework to unregister all Distribution Provider services when it’s stopped).
- unpublish any services it has published to Discovery Services before (though Distribution Provider may rely on OSGi framework to unregister all Distribution Provider services when it’s stopped). The Discovery Services will then propagate the change in availability of those particular services to other machines. This way other Distribution Providers consuming those services have a chance to gracefully react to this change.

When a new Discovery Service is activated, then it should:

- take care of already registered ServicePublication and DiscoveredServiceTracker services.

When a Discovery Service gets deactivated, then Distribution Providers should be aware that:

- They won’t be notified about changes in availability or metadata of services previously located via this Discovery Service. If a previously discovered service does change or become unavailable, then the client will start to receive errors when trying to communicate with the service. Handling of this situation is application-dependent. For example, an application may choose to stop using a particular service after a number of failures.
Any service metadata they published via this Discovery Service might still be remotely available e.g. because remote Discovery Services cached the information. So other remote Distribution Providers might still discover it and use to establish communication with the particular services. This means also that any Distribution Provider or client consuming remote services should be aware that the availability status of a service as well as the service metadata they get and use to choose appropriate services and to establish communication might be outdated and has changed already. This behavior is totally different from the local OSGi model where events regarding changes in service’s lifecycle are guaranteed to be delivered and this in a synchronous way.

When a service changes its properties at runtime, then a Distribution Provider should check:

- Whether the property change affects the remote exposure of the service e.g. property publish was set to true and if yes then act as appropriate e.g. expose the service.
- Whether it’s a service which is remotely exposed by this Distribution Provider and published to Discovery Services. In such a case the Distribution Provider should convey the change to other machines in the network and thus give them a chance to react to such a property change in a proper way e.g. the initial selection criteria may not be valid anymore. Distribution Provider may choose to republish the service with the new properties or to update the existing publication. This means also that it’s not guaranteed that updating of a service will result in an modified event on the consumer side.

It is possible for the same distribution software to be configured to expose the service over multiple protocols, or for different distribution software types to expose the same service over the same or different protocols.

### 5.7 Service Distribution using SCA Metadata

Section 5.5.3 describes how intents are used to express distribution-related requirements. For example, a service provider can express the need for confidential communications using the ‘confidentiality’ intent, and a service client can use this same intent to select a service which provides ‘confidentiality’. A deployer sees intent requirements and provisions the Distribution Software to support these (e.g. ‘confidentiality’ through an encrypted transport).

In section 5.5.3, the details of how intents are implemented are left up to the Distribution Software. This works well when the same Distribution Software is used at both ends of the communications, but when different Distribution Software systems are involved it is advantageous to use a standard mechanism for providing the additional configuration.

Distributed OSGi defines how to use SCA metadata for the concrete configuration of service distribution. It is not mandatory for a Distribution Software to support the use of this SCA metadata, but it is highly recommended as this enables greater portability and interoperability between Distribution Software types. There are a number of advantages to using the SCA metadata especially in deployments where different Distribution Software providers are involved. It enables portability of the metadata, which means a deployer does not need to learn new technologies in order to configure different Distribution Software types, and it also simplifies interoperability, where both parties involved understand the SCA metadata.

Any service using SCA metadata identifies this by setting the `osgi.remote.configuration.type` service property to “sca”, as follows:

```
osgi.remote.configuration.type = sca
```

#### 5.7.1 Bindings

SCA Bindings can be used to describe the access mechanism a client will use to call a service, or the access mechanism over which a service is made available. SCA defines a number of different binding types, for example, Web services, EJB, JMS, and also provides a mechanism for defining others.
### 5.7.1.1 Bindings on Services

A developer, or typically a deployer, can choose the bindings over which a service will be made available. This is done by setting the `osgi.remote.configuration.sca` service property to point to the detailed binding configuration. The value of this property is an array of URLs. If a URL does not have a protocol identifier, then it is assumed to be relative and refers to a resource within the contributing service’s bundle. It is recommended that bindings located in a bundle be placed in the `OSGI-INF/bindings` folder. For example,

```
osgi.remote.configuration.sca.bindings =
    OSGI-INF/bindings/beer/OrderBeerService/bindings.xml
```

Where `OSGI-INF/bindings/beer/OrderBeerService/bindings.xml` is an XML file in the bundle containing the definitions of one or more binding elements. The example below shows a document with two bindings:

```xml
<bindings>
    <binding.ws requires="soap.1_2" />
    <foocorp:binding.rmi />
</bindings>
```

Note, in this example, `binding.rmi` is a proprietary binding defined by foocorp and is therefore in the `foocorp` namespace (as denoted by the `foocorp` namespace prefix). See section 5.7.5 for information on defining new binding types.

A service configured with such a document would be made available via Web services (using soap 1.2) and RMI. The Distribution Software would expose the service at default endpoint URIs for both Web services and RMI. Bindings also allow specific endpoints URIs to be configured.

### 5.7.1.2 Bindings in Service Descriptions

A client component does not explicitly specify detailed distribution configuration for communicating with a service. The client will look up a service based on intents and business properties and then communicate with the matching services using whatever distribution configuration the target service published.

The description of the service with which the client will communicate is obtained via either Service Discovery or a static service description. Irrespective of the means by which the description is obtained, when SCA metadata is used, the description will contain properties for the detailed configuration of the target services. The following example illustrates this using static service descriptions as described in section 5.3.2.

Clients do not need to understand SCA metadata in order to be able to communicate with a service distributed using SCA metadata. For example, a service distributed using `<binding.ws />` can be called by a client using standard Web service technologies (WSDL, and soap/http).

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<service-descriptions xmlns="http://www.osgi.org/xmlns/sd/v1.0.0">
    <service-description>
        <interface name="org.acme.MyReliableService" />
        <property key="osgi.remote.interfaces">*</property>
        <property key="osgi.remote.requires.intents">exactlyOnce reliable</property>
        <property key="osgi.remote.configuration.type">sca</property>
        <property key="osgi.remote.configuration.sca.bindings">OSGI-INF/bindings/MyReliableService/bindings.xml</property>
    </service-description>
</service-descriptions>
```
The property `osgi.remote.configuration.type=sca` identifies the service description as containing SCA metadata. The detailed binding information is located at the URL, or URLs, specified in the `osgi.remote.configuration.sca.bindings` property.

In the example above, the first service description is for a service available via bindings described in `OSGI-INF/bindings/MyReliableService/bindings.xml`. Note, these bindings will be configured to support the intents "reliable" and "exactlyOnce", specified in the service description's `osgi.remote.requires.intents` property. The second service description has bindings located on a separate machine, available via http, and supporting the "confidentiality" and "integrity" intents.

In both examples, the bindings must be fully configured. In other words, they must contain sufficient information for the Distribution Software to create a proxy to talk to the real service endpoint. For example, a service description (obtained through Service Discovery or static configuration) referencing the bindings described earlier in section 5.3.2 might result in the following fully configured bindings:

```xml
<bindings>
  <binding.ws uri="http://www.beercompany.org/BeerOrderService"
             requires="soap.1_2" />
  <foocorp:binding.rmi host="www.beercompany.org" port="8099"
                       serviceName="BeerOrderService" />
</bindings>
```

In some cases it may be necessary to also include policy configuration through policySets. This is done by installing them separately as described in or referencing a definitions.xml file through a property called `osgi.remote.configuration.sca.definitions`. The format of this file is described in section 5.7.8 and the contents are the same as those described in section 5.7.4.

The combination of the service description and binding information is sufficient for a Distribution Software to create proxies for the remote service, one for Web services and one for RMI. These would be created and registered in the client framework's service registry. A client requesting a service with a filter which matches the service interface and service description properties would get one or both proxies from the service lookup.

### 5.7.2 PolicySets

PolicySets are used to define concrete policies that configure bindings. They configure a binding by providing a concrete definition of how an intent is to be fulfilled for the particular binding. PolicySets do not define their own policy language; they simply describe how to apply existing policy languages, such as WS-Policy, to bindings in support of specific intents.

The following example demonstrates this concept using the `OrderBeerService`, described earlier, and which specified intents of "confidentiality" and "reliable". A deployer might choose to configure a Web service binding to expose this service and need to configure that binding with the details of how to support "confidentiality" and "reliable". The following shows an example of how confidentiality might be configured:

```xml
<policySet name="SecurePolicy" provides="confidentiality.message"
           appliesTo="sca:binding.ws" xmlns="http://www.osoa.org/xmlns/sca/1.0">
  <!-- Configuration details for confidentiality and reliability -->
</policySet>
```
The namespace of the policySet and the way in which PolicySets are deployed into the Distribution Software is described in section 5.7.8 Error! Reference source not found..

In this example, encryption is used to define how the intent "confidentiality.message" is implemented for the Web service binding (binding.ws). The rules for determining whether intents are satisfied by bindings and policySets are the same as those used to match client requirements to service providers (see 5.5.3). For example, the intent "confidentiality" can be satisfied by a policySet or bindings which implements a qualified version, in this case "confidentiality.message".

### 5.7.3 PolicySet Attachment

SCA defines two mechanisms for attaching policySets to bindings; direct attachment where the policySets are referenced from the bindings and external attachment where the policySets state which parts of SCA they apply to. OSGi makes use of the direct attachment approach only as this is the most appropriate to OSGi's use of SCA.

Direct attachment is done by referring to the policySet from a binding definition. This can be done either through an attribute or an element. The example below shows how the “SecurePolicy” is attached to the Web service binding using an attribute.

```xml
<binding.ws policySet="foocorp:SecurePolicy" />
```

The namespace of the policySet is the targetNamespace specified in the definitions file which provided the policySet (see section 5.7.8 Error! Reference source not found.).

Note, for a service to be properly configured, all intents must be satisfied either natively by the binding (e.g. "soap" supported by binding.ws) or by the attachment of policySets.

For more details on direct policy attachment, please refer to the SCA Policy specification.

### 5.7.4 Using the Discovery Service

A Distribution Software which supports the SCA metadata may also optionally use a Discovery Service. When publishing a services over Discovery, the Distribution Software should populate the ServiceEndpointDescription with the properties described in the <service-description /> elements above. Additionally, if any service descriptions require intent or policySet definitions, then an optional property called osgi.remote.configuration.type.sca.definitions should be set, which defines a URL to these definitions. For example,

```xml
osgi.remote.configuration.type.sca.definitions = http://somemachine.com/config/definitions.xml
```
When a Distribution Software receives a published service it is responsible for fetching the additional configuration via the URLs specified in the org.remote.configuration.type.sca.xxx properties. It must then ensure it can support the bindings intents and policySet specified. If the Distribution Software is capable of supporting these, then it registers a proxy representing the remote published service.

### 5.7.5 Defining New Binding Types

In addition to defining a set of Bindings, SCA also provides a means for defining new ones. For example, a Distribution Software may wish to support rmi by defining `<foocorp:binding.rmi />`. The details of how to define new bindings are described in the section on "Defining a Binding Type" in the SCA Assembly specification. Binding Types are contributed to an SCA metadata aware Distribution Software using a definitions.xml file as described in section 5.7.8.

### 5.7.6 Defining New Intents

Section 5.5.3 describes how intents are used in the OSGi service programming model. The concepts of qualified and profile intents are introduced. SCA provides a schema for defining intents, which an SCA metadata capable Distribution Software should use.

The SCA pseudo-schema for intent definition is as follows:

```xml
<intent name="xs:string" constrains="list of QNames"
requires="list of QNames" excludes="list of QNames"?
mutuallyExclusive="boolean"?
<description> xs:string.</description> ?
<qualifier name = "xs:string" default = "xs:boolean" ?>  *
<description> xs:string.</description> ?
</qualifier>
</intent>
```

Where

- @name: specifies the name of the intent.
- @constrains: (optional) In SCA intents can apply to many specific SCA artifacts. In the OSGi usage of intents the relevant constrains value is 'sca:binding'. If this value is omitted then it is assumed that use of the intent is unrestricted.
- @requires: defines the set of all intents that the referring intent requires. This allows intents to be composed out of other intents (see section 5.5.3.12 on ‘Profile Intents’).
- @excludes: (optional) a list of intents that are incompatible with this intent. It is an error to deploy a service with incompatible intents. A Distribution Software should not distribute a service with incompatible intents.
- @mutuallyExclusive: (optional) ‘true’ signifies that the qualified intents are mutually exclusive. The default is ‘false’.
- <qualifier>: (optional) used to define a qualifier for an intent (see section 5.5.3.10 on ‘Qualified Intents’).

The intent schema is described in more detail in the SCA Policy Specification.

The following example shows a new intent called 'communicationProtection' which combines the 'confidentiality' and 'integrity' intents. Its purpose is to ensure the communications cannot be viewed or tampered with:

```xml
<intent name="communicationProtection"
constrains="binding"
requires="confidentiality integrity">
```

The document available at the URL should be a definitions XML as defined in section 5.7.8 Error! Reference source not found. below, which describes only those definitions which apply to the published service.
<description>
    Ensure that communications cannot be seen or tampered with by unauthorized personnel.
</description>

5.7.7 Intents on Bindings
Intents can be directly configured on bindings. These are considered additions to those defined in the service property `osgi.remote.requires.intents`. This mechanism enables the deployer to provide additional restrictions which only apply to a particular binding on a service. For example, the following adds "reliable" to the Web service binding:

```xml
<binding.ws requires="reliable" />
```

The details on adding intents to bindings can be found in the SCA Policy specification.

5.7.8 Definition of Intents, Binding Types and PolicySets
Intents, policySets and binding types are not specific to a particular service implementation or client and are therefore provided separate from service and service description configuration. Definitions of these are provided to a Distribution Software in a definitions.xml file. The format of the file is defined by SCA and follows the following pseudo-schema:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.osoa.org/xmlns/sca/1.0"
             targetNamespace="xs:anyURI">
    <sca:intent/>
    <sca:policySet/>
    <sca:bindingType/>
</definitions>
```

SCA defines other elements within the definitions element but these are not used in Distributed OSGi.

The targetNamespace defines the namespace in which the definitions belong, and is used to refer to them.

When a bundle containing a definitions.xml file is installed, the Distribution Software must read the file and react as follows:

- If there are services registered which refer to intents, policySets or binding types from the definitions.xml file, but which are not yet distributed, then the Distribution Software should use these definitions to distributed those services.
- If there are service descriptions (e.g. service descriptions in a `<service-descriptions />` xml file, or as result from a discovery request) which refer to something defined in the definitions.xml, then the distribution software should use these to complete the detailed configuration of the description and create a proxy to the remote service.

When a bundle containing a definitions.xml file is uninstalled the Distribution Software should react as follows:

- If there are service endpoints available for services whose configuration depended on information in the definitions.xml, then those endpoints should be removed.
- If there are service proxies to remote services whose configuration depended on information in the definitions.xml file, then those proxy services should be remove from the service registry.

5.8 Service Descriptions XML schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
<?xml version="1.0" encoding="UTF-8"?>
<annotation>
  <documentation xml:lang="en">
    This is the XML Schema for service descriptions used by Distributed OSGi. Service descriptions are used to describe remote services to a client. An extender, such as a local Discovery Service can look for service descriptions in installed bundles and then a Distribution Software can create proxies to the remote services based on the service descriptions.
  </documentation>
</annotation>
<element name="service-descriptions" type="sd:Tservice-descriptions"/>
<complexType name="Tservice-descriptions">
  <sequence>
    <element name="service-description" type="sd:Tservice-description" minOccurs="1" maxOccurs="unbounded"/>
    <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
<complexType name="Tservice-description">
  <annotation>
A Distribution Software is required to register a proxy with the interface(s) and any properties provided. If any 'intents' properties are specified then the Distribution Software should only register a proxy if it can support those intents.

```xml
<documentation xml:lang="en">
  <annotation>
    <sequence>
      <element name="provide" type="sd:Tprovide" minOccurs="1" maxOccurs="unbounded" />
      <element name="property" type="sd:Tproperty" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
    <!-- 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" />
  </annotation>
  <complexType name="Tprovide">
    <sequence>
      <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="interface" type="token" use="required"/>
  </complexType>
  <complexType name="Tproperty">
    <simpleContent>
      <extension base="string">
        <attribute name="name" type="string" use="required" />
        <attribute name="value" type="string" use="optional" />
        <attribute name="type" type="sd:Tjava-types" default="String" use="optional" />
      </extension>
    </simpleContent>
  </complexType>
  <complexType name="Tjava-types">
    <restriction base="string">
      <enumeration value="String" />
      <enumeration value="Long" />
      <enumeration value="Double" />
      <enumeration value="Float" />
      <enumeration value="Integer" />
    </restriction>
  </complexType>
</documentation>
```
Best Practices

This section is non-normative.

Handling Distributed OSGi Services

The vast majority of OSGi bundles which consume services have been designed for local service invocations. With the introduction of Distributed OSGi it is important to consider whether or not a client bundle should be able to obtain a remote service. There are two options available to avoid accidentally resolving to a remote service:

1. Ensure the Distribution Software and/or Discovery Service are configured to avoid remote services being made available in the client's service registry.

2. Install a filter hook to filter out results for remote services. RFC 126 (Service Registry Hooks) and the requirement of a Distribution Software to add the service property osgi.remote=true to all remote proxies enables this to be done.

When designing a service or client for distributed operation, there are a number of factors to consider:

1. Communications failures: ensuring proper handling of failures and uncertain outcomes, such as a request reaching a service and being acted upon, but the response never reaching the client. Consider using a reliable transport for business-critical remote communications.

2. Latency issues which may lead to requests taking longer to complete, and in some cases could result in requests arriving out of sequence. Client and service implementation design can take this into account, and if necessary, a transport can be chosen which provides the desired QoS, such as assured message ordering.

3. Invocation semantics of a remote service are likely be passByValue, whereas OSGi local services are passByReference. If a service is likely to be distributed, consider designing and implementing it to be agnostic of the call semantics as this will maximize the opportunity for the service to be re-used.

Distribution-related limitations on service interface definitions

In OSGi, service interfaces are defined using Java interfaces. When exposing a service over a remote protocol, typically such an interface is mapped to a binding-specific interface definition which is then used to advertise the interface of the service. To make sure such a mapping to a distribution protocol would work, a few things should be taken into consideration, with regard to interface definition of remote services.

So it will probably be necessary to put some constraints on the possible usage of data types in service interfaces in order to be able to expose them over remote interfaces. As an example, an interface that
has a `java.lang.Object` as an argument will probably not be allowed. The exact boundaries of the data fencing will need to be defined and it would be nice if a tool could or clear methodology could be defined that would allow the developer to test whether the interfaces at hand satisfy this requirement.

In general, the following rules should be adhered to. The Service interface should be defined in terms of:

- **Basic Types:** byte, short, int, char, long, float, double, string
- **Arrays:** of basic types or a complex type which is part of the interface
- **Complex types that are aggregations of the above.**

[do we need to add more?]

The above is sometimes referred to as ‘Data Fencing’.

Additionally, because most distribution transports use pass-by-value semantics, a developer should take care not to depend on any pass-by-reference semantics. In other words, if the caller passes an object to the Service and the Service modifies that object or makes an invocation on that object that causes a modification as a side-effect, the remote caller will not see this modification. Distributed Services should avoid such semantics.

The inverse is also an area where a developer should take care. For example, if a developer codes to a service interface assuming pass-by-value and therefore makes modifications to data which is passed in from a client or returned from a service, these modifications may become visible in the event the client and service are located in the same framework instance.

An implementation of Distributed OSGi could provide a tool that checks these constraints on your services and therefore informs the developer about the suitability for distribution.

**Discovery Service Federation and Interworking**

This section describes potential scenarios for the use of the discovery service. The implementation of a discovery service may vary depending on the distributed software system involved. These scenarios are intended to illustrate desirable use cases for using the discovery service to fulfill enterprise requirements.

![Diagram of multiple discoveries over different protocols](image)

**Figure 15 Multiple discoveries over different protocols**

The possible interaction of multiple OSGi service platforms over the discovery mechanism is shown in **Error! Reference source not found.**. Since the discovery service implementation is not specified, it is possible that multiple different protocols may be deployed simultaneously. An implementation that maintains a cache of service information over all services discovered in a network, allows for building a transitive hull over the discovery mechanism. Thus, two OSGi service platforms may discover and reference each other even though there is no common protocol used in the discovery process.
The Discovery service implementation should not publish those services that it has discovered from other discovery instances over the network. This could lead to infinite loops. However, a Discovery service implementation should answer a request over network if it is aware of a suitable instance through its cache. This may include services discovered remotely.

### Figure 16 Possible discovery service implementation

As shown in [Error! Reference source not found.], an implementation of the Discovery service interface may combine the implementation of multiple different protocols in a single bundle or provide separate bundles for different discovery protocols. This RFC does not make any assumption about the design choices.

#### Bundle organization

For an OSGi client to be able to communicate with a remote Service, it will need access to a Java interface for the service. When the Service is implemented as an OSGi bundle, the easiest way to achieve this is to put the interface of the service in a separate bundle. This bundle should then both be installed in the OSGi client environment as well as on the Service's OSGi runtime. The service's implementation will have a dependency on this bundle.

#### Proxies

On the client side the DSW is expected to create and register local endpoints for the remote services. These endpoints are typically created as proxies. In these proxies additional logic with regard to caching and load balancing may be provided as appropriate. The definition of such smart proxies is left to a separate RFC.

#### Naming convention for communication-related properties

In order to make it easier for a Distribution Provider to decide whether modification of properties reported by Discovery is related to communication or not it's recommended to use the prefix "osgi.remote" for names of communication related service properties. Service properties defined by this RFC do already follow this recommendation. Depending on the fact whether the changed property describes some communication aspects or not, Distribution Provider may decide to recreate the related service proxy or just to update its properties accordingly.
Reference Implementation

Installing Distribution Software in an OSGi platform

Both for Services and Consumers, the distribution software itself is provided as an OSGi bundle, which is installed in the OSGi platform. Any configuration for the distribution software would be provided with this bundle, and will be automatically applied when the bundle is activated.

Considered Alternatives

Alternative: using simple properties to define service remoting

This alternative was not considered viable as the simple properties approach is most likely not expressive enough.

In this alternative the remoteness of the service is simply triggered by the `remote.profile` property on the Service Declaration. `remote.profile` is a simple property that specifies on a high level how the service is remoted; possible values could be:

- SOAP/HTTP
- SOAP/HTTPS
- CORBA
- RMI
- Other

The Distribution Software picks up this value and, if it supports this kind of profile, does the appropriate thing to expose the service remotely.

On the Client-side, the remote flag is set to true, as we are dealing with a proxy. The `remote.profile` contains the value of the profile as specified in the Service declaration.

Additionally, on the client side, the `remote.url` property holds the URL of the service. In most distribution technologies a URL can be used to point to the network location where the service can be contacted. Note that this URL is only provided to the consumer for informational purposes, the client does not need to deal with the URL as all the networking is taken care of by the proxy.

On the Service-object itself the remote property should either be set to false or not be set at all. (Note that it is not mandated, but allowed, to have the `remote.profile` property as set in the DS configuration file visible on the actual service object).

Pros:

- Very simple, easy to understand for the user.
- The RFC should list a number of known profiles, which could be implemented by vendors or open source products to provide interoperability.
- Vendors can add their own proprietary profiles.
- Interoperability on the wire for standardized profiles.

Cons:

- Not very flexible. Especially w.r.t. the specification of Qualities of Service. How will you specify that SOAP/HTTP with transactions is used? SOAP/HTTP/TX? How about reliability?
has the risk of becoming unmanageable when looking at all possible combinations. How will we distinguish between different versions of a binding, e.g. SOAP 1.1 and SOAP 1.2?

- Additional configuration is always needed, which will be vendor-specific.

A variation of this approach could be taken in which transport, binding and potentially QoS information are specified in separate properties.

---

### Security Considerations

Vulnerabilities created by distributed OSGi include those in the bundles for the interfaces and proxies, and in the distributed software itself.

In the first case, distributed OSGi functionality must be implemented by trusted bundles, and deployment of distributed OSGi bundles must obtain the appropriate service permissions. Access to any resources required by the bundle or bundles also must be controlled via administrative permission. An implementation of distributed OSGi must prevent unauthorized deployment of bundles and unauthorized access to bundles and resources.

The two major security issues the DSW should address are authorized access to a service and the use of an encrypted communication protocol. When a remote service request is received, the DSW should check whether the request is authorized and also whether an encrypted protocol was used to transmit the request. If a request is not authorized for the service, the DSW should request authentication. If authentication isn’t available, the DSW should return an error stating the requester is not authorized to access the service. Similarly, if the intent attribute of confidentiality is present on the service, the DSW should check whether an encrypted communication protocol was used and return an error to the requester if it was not.

---

### Document Support

#### References

Author’s Addresses

<table>
<thead>
<tr>
<th>Name</th>
<th>Eric Newcomer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Progress Software Corp</td>
</tr>
<tr>
<td>Address</td>
<td>200 West Street, Waltham, MA 02451 USA</td>
</tr>
<tr>
<td>Voice</td>
<td>+1 781 902 8366</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:ericn@progress.com">ericn@progress.com</a>, <a href="mailto:enewcomer@gmail.com">enewcomer@gmail.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>David Bosschaert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Progress Software Corp</td>
</tr>
<tr>
<td>Address</td>
<td>The IONA Building, Shelbourne Road, Ballsbridge, Dublin 4, Ireland</td>
</tr>
<tr>
<td>Voice</td>
<td>+353 1 637 2371</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:dbosscha@progress.com">dbosscha@progress.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Tim Diekmann</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>TIBCO Software Inc.</td>
</tr>
<tr>
<td>Address</td>
<td>3303 Hillview Ave, Palo Alto, CA 94304, USA</td>
</tr>
<tr>
<td>Voice</td>
<td>+1 650 846 5521</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:tdiekman@tibco.com">tdiekman@tibco.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Graham Charters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM</td>
</tr>
<tr>
<td>Address</td>
<td>IBM United Kingdom Ltd, Hursley Park, Winchester, SO21 2JN, UK</td>
</tr>
<tr>
<td>Voice</td>
<td>+44 1962 816527</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:charters@uk.ibm.com">charters@uk.ibm.com</a></td>
</tr>
<tr>
<td>Name</td>
<td>Philipp Konradi</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Company</td>
<td>Siemens AG</td>
</tr>
<tr>
<td>Address</td>
<td>Siemens AG, Otto-Hahn-Ring 6, 81739 Munich, Germany</td>
</tr>
<tr>
<td>Voice</td>
<td>+49 (89) 636-53802</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:philipp.konradi@siemens.com">philipp.konradi@siemens.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Eoghan Glynn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Progress Software Corp</td>
</tr>
<tr>
<td>Address</td>
<td>The IONA Building, Shelbourne Road, Ballsbridge, Dublin 4, Ireland</td>
</tr>
<tr>
<td>Voice</td>
<td>+353 1 637 2439</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:eoglynn@progress.com">eoglynn@progress.com</a></td>
</tr>
</tbody>
</table>
### Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>Open CSA</td>
<td>Open Composite Services Architecture</td>
</tr>
<tr>
<td>SCA</td>
<td>Service Component Architecture</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
</tbody>
</table>
RFC 122 - Database Access

Draft

10 Pages

Abstract

This RFC describes approaches for accessing JDBC resources in an OSGi environment.
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................................................................................................................. 4

3 Problem Description................................................................................................................ 5
  3.1 DriverManager Lookup of JDBC Drivers............................................................................ 5
  3.2 JNDI Lookup of JDBC Data Sources.................................................................................. 5
  3.3 Using a Native DataSourceFactory..................................................................................... 6

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

5 Technical Solution.................................................................................................................. 6
  5.1 Existing Driver Access Approaches.................................................................................... 6
    5.1.1 DriverManager Lookup of JDBC .................................................................................. 7
    5.1.2 JNDI Lookup of JDBC Data Sources........................................................................... 7
    5.1.3 Using a Native DataSourceFactory............................................................................... 7
  5.2 Going Forward with JDBC Drivers in OSGi....................................................................... 7
  5.3 Access to JDBC Packages in OSGi..................................................................................... 8
  5.4 JDBC Resource Management............................................................................................. 9
    5.4.1 Sharing and Pooling of JDBC connections................................................................... 9
    5.4.2 Transaction Support...................................................................................................... 9

6 Security Considerations.......................................................................................................... 9

7 Document Support.................................................................................................................. 10
  7.1 References.......................................................................................................................... 10
  7.2 Author’s Address .................................................................................................................. 10
  7.3 Acronyms and Abbreviations.............................................................................................. 10
  7.4 End of Document................................................................................................................ 10
## 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 7.1.

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>Aug 14 2007</td>
<td>Initial draft with information based on RFP 77</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stuart McCulloch, Jayway Malaysia <a href="mailto:stuart.mcculloch@jayway.net">stuart.mcculloch@jayway.net</a></td>
</tr>
<tr>
<td>0.1</td>
<td>Aug 14 2007</td>
<td>Add thoughts / notes so far, as well as basic traceability matrix</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stuart McCulloch, Jayway Malaysia <a href="mailto:stuart.mcculloch@jayway.net">stuart.mcculloch@jayway.net</a></td>
</tr>
<tr>
<td>0.2</td>
<td>Jan 7 2008</td>
<td>Cover legacy use of java.sql.DriverManager utility class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stuart McCulloch, Jayway Malaysia <a href="mailto:stuart.mcculloch@jayway.net">stuart.mcculloch@jayway.net</a></td>
</tr>
<tr>
<td>0.3</td>
<td>Sep 28 2008</td>
<td>Add DataSourceFactory interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stuart McCulloch, Jayway Malaysia <a href="mailto:stuart.mcculloch@jayway.net">stuart.mcculloch@jayway.net</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mike Keith, Oracle Corp. <a href="mailto:michael.keith@oracle.com">michael.keith@oracle.com</a></td>
</tr>
<tr>
<td>0.4</td>
<td>Oct 27 2008</td>
<td>Reworked and reorganized sections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added sections on transactions and pooling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mike Keith, Oracle Corp. <a href="mailto:michael.keith@oracle.com">michael.keith@oracle.com</a></td>
</tr>
<tr>
<td>0.8</td>
<td>Jan 10 2009</td>
<td>Reorganization to put more of the descriptive background explanation in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>earlier sections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Took out sections that prescribed dynamic and declarative registration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for the DriverManager case.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mike Keith, Oracle Corp. <a href="mailto:michael.keith@oracle.com">michael.keith@oracle.com</a></td>
</tr>
<tr>
<td>0.9</td>
<td>Feb 6 2009</td>
<td>Added clarifications about the goal of this service not being a data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>source service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mike Keith, Oracle Corp. <a href="mailto:michael.keith@oracle.com">michael.keith@oracle.com</a></td>
</tr>
<tr>
<td>1.0</td>
<td>Feb 9 2009</td>
<td>Draft for vote.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mike Keith, Oracle Corp. <a href="mailto:michael.keith@oracle.com">michael.keith@oracle.com</a></td>
</tr>
</tbody>
</table>
1 Introduction

This document will only cover resource access between OSGi applications and Relational Database Management Systems (RDBMS) through JDBC. Other kinds of data sources, such as hierarchical, network-oriented or object-oriented databases, are not discussed.

The goal is to both allow applications to use existing APIs, as well as be able to benefit from the paradigms and practices that are prevalent within the OSGi framework.

This specification does not attempt to deal with the definition and registration of named and configured data sources. That would be an additional layer above this one, and is not precluded from existence by anything defined in this layer. In fact, this layer provides an appropriate level of support for such a layer to be generically implemented.

2 Application Domain

Enterprise applications are typically coupled and connected to a relational database. This RFC deals with the circumstances around database connectivity for OSGi applications in the enterprise sector.

Enterprise application bundles are often independent of each other, and many applications can reside on many OSGi frameworks, potentially 100's or even 1000's, in some form of cluster.
This document describes support for existing JDBC Specifications [4] that are both implemented by a large number of vendors and used either directly or indirectly by virtually every enterprise Java application. Ensuring compatibility with that legacy is paramount if existing applications are to be able to migrate to OSGi environments. Furthermore, as more OSGi application development tools and frameworks emerge, new enterprise applications developed specifically for OSGi will want to be able to leverage and take advantage of the features that attracted them to use OSGi to begin with. This document thus describes a standard way for JDBC to be used in future applications as well as existing ones.

3 Problem Description

OSGi is now an enterprise technology, and one of the cornerstones of enterprise applications is database connectivity. There are currently no specifications relating to the installation and use of database resource services within OSGi, and without a unified way to get hold of connections to databases, each bundle or application is left to define its own way of doing so. This RFC endeavors to define mechanisms to standardize such access within the OSGi context.

Many existing technologies leverage relational database systems to make them more object friendly. JPA and proprietary ORM products are examples of this. However, if we create a persistence implementation bundle, we also need to find a way to provide the JDBC driver to that bundle from another bundle, as it is not reasonable to assume that the persistence implementation is delivered with all possible JDBC drivers.

RBDMS's may be limited in the number of concurrent connections that they can handle. Since establishing the connection represents a cost, applications may decide to hold on to connections, keeping them open and re-using them to the highest degree possible. In OSGi, this poses a problem that bundles don't know of each others existence, and by deploying a large number of bundles that each uses its own connection, the maximum connection number may be reached fairly quickly.

If we look at OSGi deployments in clustered environments the problem becomes amplified, and connections become very limited resources.

3.1 DriverManager Lookup of JDBC Drivers

In JDBC 1.0, clients used the DriverManager utility class to register and find JDBC drivers. The DriverManager class filters out drivers that are not accessible to the caller's class loader. In OSGi this means that bundles are only able to access drivers whose main implementation class can be loaded by their bundle classloader.

In JDBC 4.0 the DriverManager class allows drivers to be discovered and loaded using either the jdbc.drivers system property to specify one or more driver classes, or the Java SE Service Provider mechanism, where JDBC-compliant drivers ship a META-INF/services/java.sql.Driver file that contains the name of the driver class.

3.2 JNDI Lookup of JDBC Data Sources

JDBC 2.0 introduced the DataSource class which was designed to work with the Java Naming and Directory Interface (JNDI) naming service, and also allowed properties to be associated with the driver instance. For mobile
devices this is the only way to register and lookup JDBC drivers, as the DriverManager class is not part of J2ME. This is also the most common way to obtain data sources in Java EE environments.

### 3.3 Using a Native DataSourceFactory

Most JDBC drivers provide a vendor-specific DataSourceFactory class that can be used to create data sources in the absence of JNDI, or when accessing the driver directly. This is less common, however, since it results in a compile-time dependency in the application to a driver-specific class.

### 4 Requirements

| RFP0077-1 | The solution MAY leverage the Service registry in OSGi. |
| RFP0077-2 | The solution MUST allow multiple connections to many databases, incl usage of JDBC drivers of different versions, within and amongst the bundles. |
| RFP0077-3 | The solution SHOULD provide a way where the JDBC driver is installed as a bundle, and the connections are instantiated declaratively in runtime. |
| RFP0077-4 | The solution MUST provide a mechanism to associate a connection description in the client with an actual connection of the provider. |
| RFP0077-5 | The solution MUST support JDBC 2.0 and later. |
| RFP0077-6 | The solution SHOULD support all types of JDBC drivers, Type 1, 2, 3 & 4. |
| RFP0077-7 | The solution MUST include a mechanism for sharing of database connections among bundles. |
| RFP0077-8 | The solution MAY require co-operative sharing of connections, i.e. acquire/release cycles. |
| RFP0077-9 | The solutions MUST allow the number of connections between the OSGi framework and the database to be limited to a relatively small number, and be manageable in runtime. |
| RFP0077-10 | The solution MUST allow for monitoring of the activity on the database connections. |
| RFP0077-11 | The solution MUST allow the creation of bundles for existing JDBC drivers. |
| RFP0077-12 | The solution SHOULD allow for inter-framework connection pools. |
| RFP0077-13 | The solution MUST not rely on the existence of the JDBC DriverManager class. |
| RFP0077-14 | The solution SHOULD provide full XA support for any resource. |

### 5 Technical Solution

#### 5.1 Existing Driver Access Approaches

Successful use of existing and traditional methods of JDBC access involves correct configuration of the client and driver. Existing methods of access are described below.
5.1.1 DriverManager Lookup of JDBC

Applications that call the DriverManager class to obtain connections will need to have the JDBC driver of their choice on their bundle classpath, or add the driver as a fragment bundle. Any approach that ensures that the class loader of the driver class is part of the class loader hierarchy of the application client will achieve the desired result.

5.1.2 JNDI Lookup of JDBC Data Sources

JNDI lookup of JDBC drivers is really just a specialized case of general JNDI lookup in OSGi, covered by RFP 84, thus when running in an environment where Java EE or JNDI is supported then the implementation will support the data source lookup in JNDI as per RFP 84. This specification does not attempt to either replace or redefine such a mechanism in OSGi, although specifications may be created in the future to do that in more OSGi-friendly ways.

5.1.3 Using a Native DataSourceFactory

Applications that rely upon the native driver-supplied factory class, or other driver-specific classes, can continue to be dependent upon the driver code by importing the driver classes that are required. They may even choose to bundle the driver with the application.

5.2 Going Forward with JDBC Drivers in OSGi

As JDBC driver implementations evolve to include built-in support for OSGi they should implement the org.osgi.service.jdbc.DataSourceFactory interface shown below and register their implementation as an OSGi service, with the driver class name stored under the "osgi.jdbc.driver" service property (JDBC_DRIVER constant in DataSourceFactory interface). This will be the way that current and future OSGi-friendly drivers will deploy into an OSGi framework.

Data sources may be managed, or perform other implementation-specific functions according to vendor-specific properties that may be offered. The provider detects the standard properties, plus any potential vendor-specific ones that clients may have supplied, and returns a DataSource instance based upon those properties.

```java
package org.osgi.service.jdbc;
import java.util.Properties;
import javax.sql.DataSource;

/**
 * DataSource providers should implement this interface and register it as an 
 * OSGi service with the JDBC driver class name in the "osgi.jdbc.driver" property. 
 */
public interface DataSourceFactory {

/**
 * Property used by JDBC driver to declare driver class when registering 
 * as a JDBC DataSourceFactory service. Clients may filter or test this 
 * property to determine if the driver is suitable, or the desired one. 
 */
public static final String JDBC_DRIVER = "osgi.jdbc.driver";

/**
 * Common property keys that DataSource clients should supply values for 
 */
```
/**
 * when calling {@link #createDataSource(Properties)}.
 */
public static final String JDBC_URL = "osgi.jdbc.url";
public static final String JDBC_USER = "osgi.jdbc.user";
public static final String JDBC_PASSWORD = "osgi.jdbc.password";

/**
* Create a new {@link DataSource} using the given properties.
* @param props properties used to configure the DataSource
* @return configured DataSource
*/
public DataSource createDataSource( Properties props );

Prospective JDBC clients look up the DataSourceFactory service, with the option to filter discovery by the driver class that they want to use. They can then invoke the factory's create method, passing in the appropriate properties, to obtain a data source that can be used to obtain JDBC connections. The sample client code below shows how a third party layer, or client application, that is written to leverage the OSGi framework can use a service tracker to get a data source.

ServiceTracker tracker =
    new ServiceTracker( bundleContext,
                        "org.osgi.service.jdbc.DataSourceFactory",
                        new DataSourceTracker( this, driverName ) );
tracker.open();

Then, in the DataSourceTracker, the addingService method might look like:

public Object addingService( ServiceReference ref ) {
    if (ref.getProperty(DataSourceFactory.JDBC_DRIVER).equals(driverName) {
        Properties props = new Properties();
        props.put( DataSourceFactory.JDBC_URL, "jdbc:derby:MyDB" );
        props.put( DataSourceFactory.JDBC_USER, "foo" );
        props.put( DataSourceFactory.JDBC_PASSWORD, "bar" );
        DataSourceFactory df = ( DataSourceFactory )
            ref.getBundle().getBundleContext().getService( ref );
        DataSource ds =  df.getDataSource( props );
        client.setDataSource( ds );
    }
}

5.3 Access to JDBC Packages in OSGi

As of JDK 1.4 the JDBC 3.0 API (both java.sql package and the extension javax.sql package) is available in the core runtime, thus standard OSGi classloading rules suffice.

No changes are required to the OSGi framework to support this.
5.4 JDBC Resource Management

JDBC drivers often manage connection resources and other associated resources to reduce allocation and cleanup costs as well as provide a level of oversight, monitoring and reuse. Driver implementations that register as OSGi services may continue to perform the kind of management activities that they might normally perform.

5.4.1 Sharing and Pooling of JDBC connections

When a driver registers as a service in the Service Registry it is provided an opportunity to offer whatever kind of data source proxy it deems necessary to support sharing of underlying connections obtained from that data source in an OSGi context. Sharing may be based upon a transaction context, a thread, a bundle, or any criteria the driver sees as providing value. Sharing parameters may be configurable, or they may be preset based on driver support.

When a driver maintains a pool of connections it vend out connections from its pool when a getConnection() method call is made on its data source, or returns them when connections are closed. When additional environment support is present, such as a Java EE runtime, then additional layers or proxies may be present, providing further pooling and resource management.

5.4.2 Transaction Support

While simple JDBC transactions are offered through the driver, higher level transaction support, including XA and 2-phase commit, are supported through the transaction manager in the framework. The driver must support XA in order to be enlisted, of course, but the responsibility for transaction initiation and completion and XAResource enlistment (using the mechanisms described in RFC 98) rests in the hands of either the application using the Connection or the container which hosts the application.

Again, in Java EE OSGi environments the transaction support will be part of the implementing container, and will offer transactional XA data source availability through JNDI lookup or dependency injection.

6 Security Considerations

Service permissions may be placed on the data source service if desired, but the real security is in the established user-level mechanisms that are already in place at the database level.
7 Document Support

7.1 References


[3]. JSR-54 "JDBC 3.0 Specification", JSR-221 "JDBC 4.0 Specification", JSR-991 "JDBC 2.1 Errata Sheet"

[4]. http://java.sun.com/j2se/1.5.0/docs/guide/jar/jar.html#Service%20Provider

7.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Stuart McCulloch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Jayway Malaysia</td>
</tr>
<tr>
<td>Address</td>
<td>B-7-6, Megan Avenue 1, 50450 Kuala Lumpur, Malaysia</td>
</tr>
<tr>
<td>Voice</td>
<td>mobile: +60 16 320 5785</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:stuart.mcculloch@jayway.net">stuart.mcculloch@jayway.net</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Mike Keith</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Oracle Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>45 O'Connor Street, Suite 400, Ottawa, Canada K1P 1A4</td>
</tr>
<tr>
<td>Voice</td>
<td>+1 613 288 4625</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:michael.keith@oracle.com">michael.keith@oracle.com</a></td>
</tr>
</tbody>
</table>

7.3 Acronyms and Abbreviations

RDBMS – Relational Database Management System

JDBC – Java Database Connectivity

JNDI – Java Naming and Directory Interface

7.4 End of Document
Abstract

The OSGi platform provides an attractive foundation for building enterprise applications. However it lacks a rich component model for declaring components within a bundle and for instantiating, configuring, assembling and decorating such components when a bundle is started. This RFC describes a set of core features required in an enterprise programming model and that are widely used outside of OSGi today when building enterprise (Java) applications. These features need to be provided on the OSGi platform for it to become a viable solution for the deployment of enterprise applications. The RFC is written in response to RFP 76
0 Document Information

0.1 Table of Contents

0 Document Information ................................................................................................................2
  0.1 Table of Contents ...............................................................................................................2
  0.2 Status..................................................................................................................................3
  0.3 Terminology and Document Conventions ..........................................................................3
  0.4 Revision History ..................................................................................................................3

1 Introduction ...............................................................................................................................9

2 Application Domain ...................................................................................................................10
  2.1 Terminology and Abbreviations ........................................................................................10

3 Problem Description ..................................................................................................................11

4 Requirements .............................................................................................................................12

5 Solution ......................................................................................................................................13
  5.1 Architectural Overview ......................................................................................................13
  5.2 Module Context Life Cycle and the Extender Bundle .......................................................15
    5.2.1 Module context creation and destruction ...............................................................15
    5.2.2 Manifest Headers for Managed Bundles ...............................................................19
    5.2.3 Module Lifecycle Events ........................................................................................20
  5.3 Declaring Module Components ........................................................................................21
    5.3.1 Naming Components .............................................................................................22
    5.3.2 Implicit Component Definitions ..............................................................................22
    5.3.3 Instantiating Components ......................................................................................23
    5.3.4 Dependencies ........................................................................................................24
    5.3.5 Component Scopes ...............................................................................................37
    5.3.6 Lifecycle .................................................................................................................38
  5.4 Interacting with the Service Registry ................................................................................40
    5.4.1 Exporting a managed component to the Service Registry ....................................40
    5.4.2 Defining References to OSGi Services ..................................................................46
    5.4.3 Dealing with service dynamics...............................................................................50
  5.5 Module Context API ..........................................................................................................53
  5.6 Namespace Extension Mechanism ..................................................................................53
    5.6.1 Date Namespace Example ....................................................................................56
  5.7 Configuration Administration Service Support ...............................................................62
    5.7.1 Property Placeholder Support................................................................................62
0.2 Status

This document specifies the Press Release process for the OSGi Alliance, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within the OSGi Alliance.

0.3 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.

0.4 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>
| 0.1 draft| Sep 13th 2007 | First draft of this RFC  
Adrian Colyer, SpringSource, adrian.colyer@springsource.com |
<p>| 0.2 draft| Nov 28th 2007 | Second draft of RFC2, incorporating design material from Spring Dynamic Modules 1.0 rc1 |</p>
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5 draft</td>
<td>Dec 21st 2007</td>
<td>Added first version of section 5.3. Defined mechanism for accessing ServiceReference objects for a collection.</td>
</tr>
<tr>
<td>0.8 draft</td>
<td>March 25th 2008</td>
<td>Addressed comments from January 2008 EEG meeting, completed section 5.3, added ModuleContextListener type.</td>
</tr>
<tr>
<td>0.9 draft</td>
<td>May 16th 2008</td>
<td>Addressed comments from John Wells and Alexandre Alves posted to EEG mailing list</td>
</tr>
<tr>
<td>0.9.1 draft</td>
<td>June 17th 2008</td>
<td>Addressed bugs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 702: don’t proxy service references</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 703 management of service references within a collection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 704 Class of a lazy init component is not reference until the component is about to be instantiated</td>
</tr>
<tr>
<td>0.9.2 draft</td>
<td>July 3rd 2008</td>
<td>Addressed bugs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 668: signature of destroy-method callback</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 672: use “mandatory” and “optional” instead of 1..N etc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 701: use ‘Map’ instead of ‘Dictionary’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 667 remove ‘legacy’ spring constructs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 695 define API for accessing components and meta-data</td>
</tr>
<tr>
<td>0.9.3 draft</td>
<td>July 7th 2008</td>
<td>Addressed bugs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 699 specification of type conversions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Completed section 5.2.3, Module Context Events</td>
</tr>
<tr>
<td>0.9.4 draft</td>
<td>September 4th 2008</td>
<td>Removed boilerplate text regarding press releases!!</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Replaced a few stray references to „bean“ with „component“</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Addressed bugs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 734 register module context as a service</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 732 diagnostic events must be asynchronous</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 744 dependency wait should be aborted on bundle stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 738 determine when and how type converters are specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 739 rethink how namespace handlers are defined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 668 signature of destroy method callback</td>
</tr>
<tr>
<td>0.9.5 draft</td>
<td>September 23rd 2008</td>
<td>Changed name to „Blueprint Service“</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Avoided over-specifying how requirement for namespace handler services should be determined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added security section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Update schema URI and location to follow OSGi conventions</td>
</tr>
<tr>
<td>Revision</td>
<td>Date</td>
<td>Comments</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0.9.6 draft</td>
<td>October 17th 2008</td>
<td>Updates from OSGi BundleFest meeting in Montpelier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 741 clarify the term Module</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 705 clarify how extender actually loads classes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 788 comments from Graham Charters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 789 comments from Rick McGuire</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 796 Bundle-Blueprint header allows directives to be specified multiple times</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 795 additional comments from Rick McGuire</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 800 clarify rules for constructor injection arguments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 805 managed-service and managed-service-factory elements should create services, not local components</td>
</tr>
<tr>
<td>0.9.7 draft</td>
<td>November 21st 2008</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>810 clarify interpretation of paths in Bundle-Blueprint header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>797 added osgi. Prefix to directives for Bundle-SymbolicName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>811 context admin events should include a bundle property</td>
<td></td>
<td></td>
</tr>
<tr>
<td>816 conflicting statement on inner component names</td>
<td></td>
<td></td>
</tr>
<tr>
<td>817 how does module context handle aliases</td>
<td></td>
<td></td>
</tr>
<tr>
<td>818 what is field and method injection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>819 ConstructorMetadataInjection not correct</td>
<td></td>
<td></td>
</tr>
<tr>
<td>820 NamedParameterSpecification class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>822,823,824,825,826.830 – miscellaneous tidy up to doc of API classes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>858 use collections in preference to arrays</td>
<td></td>
<td></td>
</tr>
<tr>
<td>861 clarify placement of different element types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>882 constructor resolution rules and factory methods</td>
<td></td>
<td></td>
</tr>
<tr>
<td>888 schema for property types don’t agree with specification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>910 remove alias concept and local attribute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>796 remove osgi. prefix from blueprint directives</td>
<td></td>
<td></td>
</tr>
<tr>
<td>834 reference to wildcard *.xml still remaining</td>
<td></td>
<td></td>
</tr>
<tr>
<td>846 error in grammar of manifest header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>848 inconsistent use of id and name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>841 osgi. prefix for scope types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>806 signature of ModuleContextListener methods</td>
<td></td>
<td></td>
</tr>
<tr>
<td>859 ServiceExportComponentMetadata should return a dictionary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>833 conflicting statements about bundle state</td>
<td></td>
<td></td>
</tr>
<tr>
<td>835 Inconsistent definition of wait for dependencies semantics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>839 ModuleContext.getComponent and lazy initialization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>840 ModuleContext.getComponent and scopes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>842 make registration/unregistration methods required on listener</td>
<td></td>
<td></td>
</tr>
<tr>
<td>844 inconsistent timeout units</td>
<td></td>
<td></td>
</tr>
<tr>
<td>845 reference to „ComponentRegistry“ should be „ComponentDefinitionRegistry“</td>
<td></td>
<td></td>
</tr>
<tr>
<td>847 EventAdmin events not consistently setting properties</td>
<td></td>
<td></td>
</tr>
<tr>
<td>860 description element not defined</td>
<td></td>
<td></td>
</tr>
<tr>
<td>869 getClassName javadoc updated for factory-component case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>870 need to allow both type and index on parameter specification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>871 order of ConstructorInjectionMetadata.getParameterSpecifications must be specified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>875 is there a defined order for property injection?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.9.8</td>
<td>December 12th 2008</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td></td>
</tr>
</tbody>
</table>
| • 815 remove references to multiple extenders  
• 970 fixed a couple of typos  
• 928 clarified documentation of wait-for-dependencies and fixed bad attribute value for availability in schema  
• 862 LAZY_ACTIVATION and timeouts  
• 963 Described how listeners participate in dependency cycles  
• 942 ReferenceNameValue is not a suitable value option for ServiceExportComponentMetadata  
• 867 array type conversions  
• 836 key-type and value-type attributes for collections  
• 896 access to converters |

<table>
<thead>
<tr>
<th>0.9.9</th>
<th>January 7th 2009</th>
</tr>
</thead>
</table>
| • 862 qualified that mandatory service timeouts are based on receipt of LAZY_ACTIVATION event for bundles with lazy activation.  
• 1003 assorted typos fixed  
• 1010 circular logic in lazy activation description  
• 973 handling of multiple service listener methods  
• 974 use Map not Dictionary in APIs  
• 986 removed errant references to local= attribute  
• 987 example of <ref> tag usage incorrect  
• 993 adding numbered headings to section 5.4.3.1  
• 996 reference to Map in javadoc of CollectionBasedServiceReferenceMetadata deleted  
• 998 schema mixes identifiedType and TidentifiedType  
• 994 define scope of inner components |
1.0 February 11th 2009

- 985 circular references
- 1017 is lazy-init implied for prototype components?
- 1018 interaction of depends-on and prototype scope
- 1019 destroy-method calls for bundle-scoped components
- 1020 do bundle scoped instances persist until module context destroy?
- 1028 added example for case of unsatisfied optional dependency
- 1035 fixed error in registration listener example
- 1042 can <interfaces> element be used with ref. Collections?
- 1043 should ref-set and ref-list have a timeout attribute
- 1044 provide more explicit schema definition for listeners
- 1045 document ref= attribute for listener tags
- 1047 WAITING events for blocked service invocations
- 1048 fixed bad reference to „osgi“ element in section 5.4.3.3
- 1051 use blueprint-compendium not blueprint/compendium in xmlns
- 1054 allow limitation on final classes in section 5.4.2.1
- 1055 added constants for EventAdmin topics
- 1070 clarified timing of CREATING event for bundles with LAZY_ACTIVATION
- 1071 fixed broken cross-reference in section 5.5
- 1034 missing / in closing element tags
- 1074 incorrect schema definition for default-properties element
- 1077 replaced ModuleContextAware with moduleContext component
- 1076 combine all implicit component definitions into one section
- 1078 return value of NamespaceHandler.getSchemaLocation
- 1080 confusing sentence regarding managed-properties and constructor-injection in section 5.7.3
- 992 Clarify usage of inner components
- 1053 added attribute to specify desired member type for ref-set and ref-list
- 1085 rename config-properties to cm-properties
- 1084 handling of container managed properties on update
- 1081 Reference state during bind/unbind callbacks
- 1046 Clarify method signatures for bind and unbind
- 1083 ListValue, SetValue, MapValue made consistent with other Value types
- 1036 clarify the meaning of „request“ for prototype components
- 1049 simplify sorting of reference collections
- 1033 incomplete set of metadata Value types
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1090 access to pid for a managed-component</td>
<td>1092 there is no requirement to use a fixed BSN</td>
</tr>
<tr>
<td>1093 old reference to managed-service element replaced with managed-properties.</td>
<td>1095 clarified behaviour of property-placeholder evaluation</td>
</tr>
<tr>
<td>976 registering services with array property types</td>
<td>975 array type specification</td>
</tr>
<tr>
<td>1082 why is ParserContext.getEnclosingMetadata needed?</td>
<td></td>
</tr>
</tbody>
</table>

## 1 Introduction

In 2006 SpringSource (formerly known as Interface21), the company behind the Spring Framework ("Spring"), identified a complementary relationship between the application assembly and configuration features supported by Spring, and the modularity and versioning features of OSGi. Spring is primarily used to build enterprise Java applications. In this marketplace there is a need for a solution to versioning, simultaneous deployment of more than one version of a library, a better basis for dividing an application into modules, and a more flexible runtime and deployment model. OSGi provides a proven solution to these problems. The question became, how can enterprise application developers take advantage of OSGi (build enterprise applications as a set of OSGi bundles) when developing Spring applications?

In response to this challenge, the Spring Dynamic Modules project was born (formerly known as the Spring-OSGi project). Spring Dynamic Modules enables the use of Spring to configure both the internals of a bundle and also references between bundles. Even with little promotion the project quickly gathered a lot of attention. As of September 2007 there are over 800 users subscribed to the project’s active discussion group. Enterprise developers have responded extremely positively to the direction being taken by the project. The Spring Dynamic Modules project is led by SpringSource, with committers from Oracle and BEA also active. The design of Spring Dynamic Modules has been influenced by discussion (both face-to-face and in the discussion group) with key personnel in the OSGi Alliance and from the equinox, Felix, and Knopflerfish OSGi implementations.

The strong interest in the Spring-OSGi project demonstrates that the enterprise Java market is attracted to the OSGi platform, and that the set of capabilities offered by Spring Dynamic Modules represent important additions to the OSGi platform. At the OSGi Enterprise Expert Group requirements meeting held in Dublin in January 2007 a working group was formed to create an RFP for adding these capabilities to OSGi. The resulting RFP, RFP 76, was accepted by the OSGi Alliance, and this RFC is written in response to the requirements documented there.
2 Application Domain

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 “front-office” 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.

2.1 Terminology and Abbreviations

1. Inversion of Control: a pattern in which a framework is in control of the flow of execution, and invokes user-code at appropriate points in the processing.
2. Dependency Injection: a form of inversion of control in which a framework is responsible for providing a component instance with its configuration values and with references to any collaborators it needs (instead of the component looking these up).
3. Aspect-oriented programming (AOP): a programming paradigm in which types known as “aspects” provide modular implementations of features that cut across many parts of an application. AspectJ is the best known AOP implementation.
4. Application Context: a Spring container that instantiates, configures, assembles and decorates component instances known as beans, also used to refer to an instance of a Spring container.
5. Bean: a component in a Spring application context
6. JMS: Java Messaging Service
7. JPA: Java Persistence API
8. Java Servlets: Java standard for serving web requests
9. EJB: Enterprise JavaBeans component model defined by the Java Platform, Enterprise Edition
10. JTA: Java Transaction API
11. JSF: JavaServer Faces, component model for web user interfaces
12. JAX-WS: Java API for XML-based web services
13. Module context: a container instance responsible for instantiating, configuring, and managing components within a module. A bundle has 0..1 module contexts associated with it.
14. Managed component: a component instantiated and configured by a module context.

3 Problem Description

Enterprise application developers working with technologies such as those described in chapter 2 would like to be able to take advantage of the OSGi platform. The core features of enterprise programming models previously described must be retained for enterprise applications deployed in OSGi. The current OSGi specifications are lacking in the following areas with respect to this requirement:

- There is no defined component model for the internal content of a bundle. Declarative Services only supports the declaration of components that are publicly exposed.
- The configuration (property injection) and assembly (collaborator injection) support is very basic compared to the functionality offered by frameworks such as Spring.
- There is no model for declarative specification of services that cut across several components (aspects or aspect-like functionality)
- Components that interact with the OSGi runtime frequently need to depend on OSGi APIs, meaning that unit testing outside of an OSGi runtime is problematic
- The set of types and resources visible from the context class loader is unspecified. The context class loader is heavily used in enterprise application libraries
- Better tolerance of the dynamic aspects of the OSGi platform is required. The programming model should make it easy to deal with services that may come and go, and with collections of such services, via simple abstractions such as an injecting a constant reference to an object implementing a service interface, or to a managed collection of such objects. See the description of osgi:reference in the Spring Dynamic Modules specification for an example of the level of support required here.

Providing these capabilities on the OSGi platform will facilitate the adoption of OSGi as a deployment platform for enterprise applications. This should be done in a manner that is familiar to enterprise Java developers, taking into account the unique requirements of the OSGi platform. The benefits also extend to other (non-enterprise) OSGi applications that will gain the ability to write simpler, more testable bundles backed by a strong component model.
4 Requirements

1. The solution MUST enable the instantiation and configuration of components inside a bundle based on metadata provided by the bundle developer.
2. The solution SHOULD NOT require any special bundle activator or other code to be written inside the bundle in order to have components instantiated and configured.
3. The solution MAY choose to provide an extender bundle that is responsible for instantiating and configuring components inside a bundle with component metadata, when such bundles are started.
4. The solution SHOULD enable the creation of components inside a bundle to be deferred until the dependencies of those components are satisfied.
5. The solution MUST provide guarantees about the set of resources and types visible from the context class loader during both bundle initialization and when operations are invoked on services.
6. The solution MAY provide a means for components to obtain OSGi contextual information (such as access to a BundleContext) without requiring the programmer to depend on any OSGi “lookup” APIs. This is required so that components may be unit tested outside of an OSGi runtime.
7. The solution MUST provide a mechanism for a bundle component to be optionally exported as an OSGi service. It MAY provide scope management for exported service (for example, a unique service instance for each requesting bundle).
8. The solution MUST provide a mechanism for injecting a reference to an OSGi service into a bundle component. It SHOULD provide a constant service reference that the receiving component can use even if the target service backing the reference is changed at run time.
9. The solution MUST provide a mechanism for injecting a reference to a set of OSGi services into a bundle component. It SHOULD provide access to the matching OSGi services via a constant service reference that the receiving component can use even if the target services backing the reference change at run time.
10. The solution MUST provide a mechanism for service clients obtaining references as described to be notified when a backing target service is bound or unbound.
11. The solution SHOULD tolerate services in use being unregistered and support transparent rebinding to alternate services if so configured.
12. The solution SHOULD support configuration of bundle components with configuration data sourced from the OSGi Configuration Admin service. It SHOULD support re-injection of configuration value if configuration information is changed via the Configuration Admin service after the bundle components have been initially instantiated and configured.
13. The solution SHOULD provide a rich set of instantiation, configuration, assembly, and decoration options for components, compatible with that expected by enterprise programmers used to working with containers such as Spring.
14. The solution SHOULD allow multiple component instances to be created dynamically at runtime.
15. The solution SHOULD present a design familiar to enterprise Java developers.
16. The solution MUST enable bundles configured using the component model to co-exist with bundles using Declarative Services
17. The solution MUST define capabilities available on the OSGi minimum execution environment
18. The solution MAY define enhanced capabilities available on other execution environments, as long as there is a strict subset/superset relationship between the features offered in less capable execution environments and the features offered in more capable execution environments.

5 Solution

5.1 Architectural Overview

The runtime components to be created for a bundle, together with their configuration and assembly information, are specified declaratively in one or more configuration files contained within the bundle. This information is used at runtime to instantitate and configure the required components when the bundle is started. A bundle with such information present is known as a managed bundle. The configuration and assembly information can be regarded as a “blueprint” for creating the runtime components of the managed bundle, hence this RFC is named “Blueprint Service”.

An extender bundle is responsible for observing the life cycle of such bundles. When a bundle is started, the extender creates a module context1 for that bundle from its blueprint by processing the configuration files and instantiating, configuring, and assembling the components specified there. The module context is a lightweight container that manages the created components, known as managed components. When a managed bundle is stopped, the extender shuts down the module context, which causes the managed components within the context to be cleanly destroyed.

The declarative configuration for a bundle may also specify that certain of the bundle's managed components are to be exported as services in the OSGi service registry. In addition, it is possible to declare that a bundle component depends on a service or set of services obtained via the service registry, and to have those services dependency injected into the bundle component.

The solution therefore supports an application architecture in which modules are implemented as OSGi bundles with a module blueprint (the configuration information) and a runtime module context created from that blueprint. Modules are peers which interact via the service registry.

Figure 1 below provides a pictorial overview of the solution. Note that there is no reason an “infrastructure” bundle cannot also contain configuration information and have a module context automatically managed for it. From the perspective of the extender, all bundles are equal.

---

1 It is tempting to call this a “bundle context”, but that could cause confusion with the BundleContext interface. In Spring this concept is know as an “application context”.
The remainder of this section is structured as follows:

- Section 5.2 explains the relationship between bundles and module contexts and the role of the extender bundle
- Section 5.3 defines the configuration support for declaring components within a module context
- Section 5.4 defines how to export managed components as services in the service registry, and how to import references to services obtained via the registry
- Section 5.5 defines the interaction with the OSGi configuration administration service
5.2 Module Context Life Cycle and the Extender Bundle

5.2.1 Module context creation and destruction

A Blueprint Service implementation must provide an extender bundle which manages the lifecycle of module contexts. This bundle is responsible for creating the module contexts for managed bundles (every ACTIVE managed bundle has one and only one associated module context). When the extender bundle is installed and started it looks for any existing managed bundles that are already in the ACTIVE state and creates module contexts on their behalf. In addition, it listens for bundle STARTED events and automatically creates a module context for any managed bundle that is subsequently started.

The extender bundle creates module contexts asynchronously. This behavior ensures that starting an OSGi Service Platform is fast and that bundles with service inter-dependencies do not cause deadlock on startup. A managed bundle may therefore transition to the ACTIVE state before its module context has been created. If module context creation fails for any reason then a context creation failure event will be published. The bundle remains in the ACTIVE state. There will be no services exported to the registry from the module context in this scenario.

When a module context has been successfully created, an instance of org.osgi.service.blueprint.context.ModuleContext describing the context is published in the service registry under the org.osgi.service.blueprint.context.ModuleContext interface. This service has a service property osgi.blueprint.context.symbolicname set to the bundle symbolic name of the bundle for which the context has been created. In addition a service property osgi.blueprint.context.version is set to the bundle version of the bundle for which the context has been created. This means that the symbolicname and version service properties are technically redundant because the same information can be determined by querying the ServiceRegistration object for the service to determine the bundle that published it. However, these properties are required to be published so that a service tracker can easily be given a filter expression that will wait for the publication of a module context service for a given bundle.

If a component to be created for a module context declares a mandatory dependency on the availability of certain OSGi services (see Section 5.4) then creation of the module context is blocked until the mandatory dependency can be satisfied through matching services available in the OSGi service registry. Since a service may come and go at any moment in an OSGi environment, this behavior only guarantees that all mandatory services were available at the moment creation of the module context began. One or more services may subsequently become unavailable again during the process of module context creation. Section 5.4 describes what happens when a mandatory service reference becomes unsatisfied.

A timeout applies to the wait for mandatory dependencies to be satisfied. If mandatory dependencies have not been satisfied before the timeout, then context creation fails. By default the timeout is set to 5 minutes, but this value can be configured using the timeout directive. See below for more information on manifest header entries and the available directives.

It is possible to change the module context creation semantics so that application context creation fails if all mandatory services are not immediately available upon startup.

When a managed bundle is stopped, the module context created for it is automatically destroyed. All services exported by the bundle will be unregistered (removed from the service registry) and any managed components within the module context that have specified destroy callbacks will have these invoked. Destruction of the module context for a STOPPING bundle happens synchronously. The context is guaranteed to be destroyed by the time the bundle transitions to the RESOLVED state.

If a managed bundle that has been stopped is subsequently re-started, a new module context will be created for it.

If the extender bundle is stopped, then all the module contexts created by the extender will be destroyed. Module contexts are shutdown in the following order:
1. Module contexts that do not export any services, or that export services that are not currently referenced, are shutdown in reverse order of bundle id. (Most recently installed bundles have their module contexts shutdown first).

2. Shutting down the module contexts in step (1) may have released service references these contexts were holding such that there are now additional module contexts that can be shutdown. If so, repeat step 1 again.

3. If there are no more active module contexts, we have finished. If there are active module contexts then there must be a cyclic dependency of references. The circle is broken by determining the highest ranking service exported by each context: the bundle with the lowest ranking service in this set (or in the event of a tie, the highest service id), is shut down. Repeat from step (1).

5.2.1.1 Lazy Activation

Since the OSGi R4.1 specification, a bundle may be marked as having lazy activation. Such a bundle issues a LAZY_ACTIVATION event when it is started, and then waits until a class is loaded from the bundle. On the first class load the STARTING event is issued and the Bundle Activator is invoked. When the activator returns the STARTED event is issued, and the bundle proceeds to the ACTIVE state.

Module context instantiation is triggered when the STARTED event is issued for a given bundle. Therefore the module context for a lazily activated bundle will not be created until a class has been loaded from that bundle.

An additional consideration is the visibility of services registered by a module context when it is started. For a bundle that has been lazily activated but not yet started no module context has been created and hence the services are not yet published. Instead, placeholder services (ServiceFactory instances) are registered on the LAZY_ACTIVATION event so that these services can still be discovered and invoked by other bundles. The invocation of getService on a placeholder service factory also triggers the lazy bundle to be fully started. See section 5.4.1.6 for more details.
5.2.1.2 Module Life Cycle Summary

LAZY_ACTIVATION event

STARTED event for bundle with eager activation

state = CREATING event CREATING

mandatory service refs?

yes

services available?

no

state = WAITING event WAITING

wait for timeout or service availability

yes

services available?

no (timeout)

lazy activation?

yes

register proxy services

wait for trigger

to instantiation flow

to failure flow
5.2.2 Manifest Headers for Managed Bundles

The extender recognizes a bundle as a managed bundle and will create an associated module context when the bundle is started if one or both of the following conditions is true:

- The bundle path contains a folder OSGI-INF/blueprint with one or more files in that folder with a '.xml' extension.
- META-INF/MANIFEST.MF contains a manifest header Bundle-Blueprint.

In the absence of the Bundle-Blueprint header the extender expects every "*.xml" file in the OSGI-INF/blueprint folder to be a valid module context configuration file. A single module context is constructed from this set of files. Supporting multiple configuration files gives developers the option to separate out parts of their configuration. For example, security related definitions may be grouped together into a file of their own, or local component definitions and service import/export definitions split into separate files. From the perspective of the extender bundle, the choice of one or many configuration files makes no difference.

The Bundle-Blueprint manifest header may be used to specify an alternate set of configuration files. The resource paths are treated as relative resource paths and resolve to entries defined in the bundle and the set of attached fragments. When the Bundle-Blueprint header defines at least one configuration file location, any files in OSGI-INF/blueprint are ignored unless directly referenced from the Bundle-Blueprint header.

The syntax for the Bundle-Blueprint header value is:

```
header ::= clause (',' clause)*
clause ::= path (';' path)* (';' parameter)*
```

Parameter can be either a directive or an attribute. This syntax is consistent with the OSGi Service Platform common header syntax defined in section 3.2.3 of the OSGi Service Platform Core Specification. The blueprint service itself defines no standard directives or attributes, but implementations of this specification may make use of this extension mechanism should they wish to do so.

For example, the manifest entry:

```
Bundle-Blueprint: config/account-data-context.xml, config/account-security-context.xml
```

will cause a module context to be instantiated using the configuration found in the files account-data-context.xml and account-security-context.xml in the bundle jar file.

Two directives are defined by the blueprint service specification to control the manner in which module context creation occurs. These directives are applied to the Bundle-SymbolicName header.

- `blueprint.wait-for-dependencies` (true|false)
  
  controls whether or not module context creation should wait for any mandatory service dependencies to be satisfied before proceeding (the default), or proceed immediately without waiting if dependencies are not satisfied upon startup.

  For example:

  ```
  Bundle-SymbolicName: org.osgi.foobar;blueprint.wait-for-dependencies:=false
  ```

  When wait-for-dependencies is false, context creation will begin immediately even if dependencies are not satisfied. For unsatisfied mandatory service references this means that the module context will proceed to the state it would be in had any such mandatory service references been available when module context creation began, but had subsequently become unavailable. The bundle will be in the ACTIVE state and any exported services (service components) that depend on the unsatisfied mandatory service references will
not be registered in the service registry until such time as the mandatory service reference becomes satisfied. Exported services that do not depend on the unsatisfied mandatory service reference will be registered in the service registry as usual. Service components that depend on the mandatory service reference will be injected with a service object that may not be backed by an actual service in the registry initially. See section 5.4 for more details.

- **blueprint.timeout (300000)**

  the time to wait (in milliseconds) for mandatory dependencies to be satisfied before giving up and failing module context creation. This setting is ignored if blueprint.wait-for-dependencies:=false is specified. The default is 5 minutes (300000 milliseconds).

For example:

```text
Bundle-SymbolicName: org.osgi.foobar;blueprint.timeout:=60000
```

Creates a module context that waits up to 1 minute (60 seconds) for its mandatory dependencies to appear.

### 5.2.3 Module Lifecycle Events

When a module context has been successfully created, the extender bundle must invoke the "contextCreated" operation of any registered services advertising support for the org.osgi.service.blueprint.context.ModuleContextListener interface. Only services with a compatible version of the interface will be invoked.

When creation of a module context fails for any reason, then the extender bundle must invoke the "contextCreationFailed" operation of any registered services advertising support for the org.osgi.service.blueprint.context.ModuleContextListener interface. Only services with a compatible version of the interface will be invoked.

Finer-grained information about the creation of module contexts is available if an EventAdmin service is available. When an EventAdmin service is available, events are published on the following topics:

- `org/osgi/service/blueprint/context/CREATING` – the extender has started to create a module context
- `org/osgi/service/blueprint/context/CREATED` – a module context has been successfully created
- `org/osgi/service/blueprint/context/DESTROYING` – the extender is destroying a module context
- `org/osgi/service/blueprint/context/DESTROYED` – a module context has been destroyed
- `org/osgi/service/blueprint/context/WAITING` – creation of a module context is waiting on the availability of a mandatory service, or a service invocation is waiting on the availability of a suitable backing service (see section 5.4.3).
- `org/osgi/service/blueprint/context/FAILURE` – creation of a module context has failed

All events are to be delivered asynchronously using the postEvent operation.

For each event the following properties are published:

- "bundle.symbolicName" (String) the symbolic name of the bundle for which the context is being created / destroyed.
- "bundle.id" (Long) the id of the bundle for which the context is being created / destroyed
- "bundle" (Bundle) the Bundle object of the bundle for which the context is being created or destroyed
- "bundle.version" (Version) the version of the bundle for which the context is being created / destroyed
- "timestamp" (Long) the time when the event occurred
- "extender.bundle" (Bundle) the Bundle object of the extender that is processing the context
• "extender.bundle.id" (Long) the id of the extender bundle that is processing the context
• "extender.bundle.symbolicName" (String) the symbolic name of the extender bundle that is processing the context

In addition for a FAILURE event the "exception" property contains a Throwable detailing the failure cause. For a WAITING event, the "service.objectClass" (String[]) property details the interface type(s) of the service that the context is waiting on, and the "service.Filter" (String) property details the filter (if any). If the root cause of a FAILURE event is a timeout for a service wait, then the "service.objectClass" and "service.Filter" properties are also required to be present as part of the FAILURE event and these provide information about the service on which the wait failed. If the timeout occurred as a result of multiple services being unavailable then it is unspecified which of the unsatisfied references is reported in this manner.

The property names for module context events may be conveniently referenced using the constants defined in the org.osgi.service.event.EventConstants and org.osgi.service.blueprint.context.ModuleContextEventConstants interfaces.

A WAITING event is issued when a mandatory service is unavailable during context creation. An implementation may deliver one or more WAITING events for the same unsatisfied service reference before either the reference is satisfied, creation times out, or the bundle for which the context is being created is STOPPED.

If the bundle for which the module context is currently being created is STOPPED during the creation process then the DESTROYING and DESTROYED events will be published and any pending waits for that bundle (as notified by WAITING events) can be assumed to be cancelled.

### 5.3 Declaring Module Components

A module context configuration file contains component definitions using XML declarations from the osgi namespace (see section 5.7). As described in section 5.2.1.1, the complete blueprint for a module context may be comprised of definitions from multiple configuration files. There is at most one module context created for any given ACTIVE bundle. The module context container manages the lifecycle of these components. The basic structure of a configuration file is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

  <description>
    Optional description for the blueprint defined in this file.
  </description>

  <component id="..." class="...">
    <!-- collaborators and configuration for this component go here -->
  </component>

  <component id="..." class="...">
    <!-- collaborators and configuration for this component go here -->
  </component>

  <!-- more component definitions go here... -->

</components>
```

If an error occurs during the processing of blueprint configuration files then creation of the context will fail with a ComponentDefinitionException (accessible via the "exception" property of the FAILURE event.)
A module container manages one or more components. These components are created using the configuration metadata (blueprint) that has been supplied to the container. Component definitions contain the following metadata:

- a package-qualified class name: typically this is the actual implementation class of the component being defined.
- component behavioral configuration elements, which state how the component should behave in the container (scope, lifecycle callbacks, and so forth).
- references to other components which are needed for the component to do its work; these references are also called collaborators or dependencies.
- other configuration settings to set in the newly created object. An example would be the number of connections to use in a component that manages a connection pool, or the size limit of the pool.

### 5.3.1 Naming Components

Every component has at most one id (also called an identifier, or name; these terms refer to the same thing). Component ids must be unique within a module.

Please note that you are not required to supply an id for a component. If no id is supplied explicitly, the container will generate a unique id for that component unless the component is an inner component (see 5.3.4.1.4 Inner Components on page 29). Inner components are always anonymous.

### 5.3.2 Implicit Component Definitions

A module context contains a number of implicitly defined components with well-known names. A user defined component with the same name as one of these will override the implicit component definition, but it is not recommended to define such components.

#### 5.3.2.1 ModuleContext component

The ModuleContext interface provides access to the component objects within the module context and to metadata describing the components within the context. A module context contains an implicitly defined component called 'moduleContext' which supports the ModuleContext interface.

See section 5.8 for a description of the ModuleContext and ModuleContextAware interfaces.

The ModuleContext interface is useful for component lookup by name (where for example a component name has been specified via an external configuration mechanism such as the configuration admin service, or the component being looked up has a narrower scope than the component looking it up). It is also commonly used for introspection on the module context for management and other purposes.

Care should be taken with the use of the ModuleContext interface. In general the lookup pattern is discouraged and regular dependency injection is to be preferred where possible. When using the ModuleContext interface from within the module context it represents, the types of the components returned by the ModuleContext operations are guaranteed to be compatible with the caller. When using a ModuleContext interface from outside of the module context it represents (for example, after obtaining a reference to a ModuleContext via the service registry) then there is no guarantee of type compatibility or even visibility between the versions of the types of the returned components, and the versions of the types visible to the caller. Care must therefore be taken if casting the return value of ModuleContext.getComponent to a more specific type.

#### 5.3.2.2 Bundle and BundleContext components

A module context contains two implicitly defined components, with ids “bundle” and “bundleContext” respectively. The “bundle” component is of type org.osgi.framework.Bundle and represents the bundle with which the module context is associated. The “bundleContext” component is of type org.osgi.framework.BundleContext and is the BundleContext object for the bundle with which the module context is associated.
5.3.2.3 ConversionService component

A module context contains an implicitly defined component with id "conversionService" of type org.osgi.service.blueprint.convert.ConversionService, see section 5.3.4.1.6 for more information.

5.3.3 Instantiating Components

You can specify the type (or class) of object that is to be instantiated using the 'class' attribute of the <component/> element. The class element specifies the class of the component to be constructed in the common case where the container itself directly creates the component by calling its constructor reflectively (somewhat equivalent to Java code using the 'new' operator). In the less common case where the container invokes a static, factory method on a class to create the component, the class property specifies the actual class containing the static factory method that is to be invoked to create the object (the type of the object returned from the invocation of the static factory method may be the same class or another class entirely, it doesn't matter).

Any types referenced in configuration elements and that need to be loaded as part of module context creation must be visible to the bundle defining the blueprint. All such types are loaded from the class space of the blueprint-defining bundle.

5.3.3.1 Instantiation using a constructor

When creating a component using the constructor approach, the class being created does not need to implement any specific interfaces or be coded in a specific fashion. Just specifying the component class should be enough. However, depending on what type of IoC you are going to use for that specific component, you may need a default (empty) constructor – this is required for “setter” injection.

You can specify your component class like so:

```
<component id="exampleComp" class="examples.Example"/>
<component name="anotherExample" class="examples.ExampleTwo"/>
```

The mechanism for supplying arguments to the constructor (if required), or setting properties of the object instance after it has been constructed, is described shortly.

5.3.3.2 Instantiation using a static factory method

When defining a component which is to be created using a static factory method, along with the class attribute which specifies the class containing the static factory method, another attribute named factory-method is needed to specify the name of the factory method itself. The container expects to be able to call this method (with an optional list of arguments as described later) and get back a live object, which from that point on is treated as if it had been created normally via a constructor. One use for such a component definition is to call static factories in legacy code.

The following example shows a component definition which specifies that the component is to be created by calling a factory-method. Note that the definition does not specify the type (class) of the returned object, only the class containing the factory method. In this example, the createInstance() method must be a static method.

```
<component id="exampleComponent" class="examples.ExampleComponent2"
           factory-method="createInstance"/>
```

The mechanism for supplying (optional) arguments to the factory method, or setting properties of the object instance after it has been returned from the factory, will be described shortly.

5.3.3.3 Instantiation using an instance factory method

In a fashion similar to instantiation via a static factory method, instantiation using an instance factory method is where a non-static method of an existing component from the container is invoked to create a new component. To
use this mechanism, the 'class' attribute must be left empty, and the 'factory-component' attribute must specify the name of a component\(^2\) in the container that contains the instance method that is to be invoked to create the object. The name of the factory method itself must be set using the 'factory-method' attribute.

```xml
<!-- the factory component, which contains a method called createService() -->
<component id="serviceLocator" class="com.foo.DefaultServiceLocator">
   <!-- inject any dependencies required by this locator component -->
</component>

<!-- the component to be created via the factory component -->
<component id="exampleComponent"
   factory-component="serviceLocator"
   factory-method="createService"/>
```

### 5.3.4 Dependencies

A typical module is not made up of a single object (or component). Even the simplest of modules will no doubt have at least a handful of objects that work together. This next section explains how you go from defining a number of component definitions that stand alone to a fully realized module where objects work (or collaborate) together to achieve some goal.

#### 5.3.4.1 Injecting Dependencies

The basic principle behind Dependency Injection (DI) is that objects define their dependencies (that is to say the other objects they work with) only through constructor arguments, arguments to a factory method, or properties which are set on the object instance after it has been constructed or returned from a factory method. Then, it is the job of the container to actually inject those dependencies when it creates the component. This is fundamentally the inverse, hence the name Inversion of Control (IoC), of the component itself being in control of instantiating or locating its dependencies on its own using direct construction of classes, or something like the Service Locator pattern.

##### 5.3.4.1.1 Constructor Injection

Constructor-based DI is achieved by invoking a constructor with a number of arguments, each representing a dependency. Additionally, calling a factory method with specific arguments to construct the component can be considered almost equivalent, and the rest of this text will consider arguments to a constructor and arguments to a factory method similarly. Specifically, the `<constructor-arg>` element can also be used to specify arguments to factory methods, and the disambiguation rules for constructors also apply to disambiguation of overloaded factory methods.

The SimpleMovieLister class below an example of a class that could only be dependency injected using constructor injection. Notice that there is nothing special about this class.

```java
public class SimpleMovieLister {
   // the SimpleMovieLister has a dependency on a MovieFinder
   private MovieFinder movieFinder;
}
```

\(^2\) There are several ways to declare a named component. For example, a component may be declared using the `<component>` element, and also using the `<reference>` element defined in section 5.4.2. Namespace elements introduced by namespace handlers (see section 5.6) may provide alternate mechanisms for defining named components too.
// a constructor so that the container can 'inject' a MovieFinder
public SimpleMovieLister(MovieFinder movieFinder) {
    this.movieFinder = movieFinder;
}
// business logic that actually 'uses' the injected MovieFinder is omitted...
}

Constructor arguments are specified using the <constructor-arg> element. For example, an instance of the SimpleMovieLister class could be configured as follows:

<component name="movieLister" class="SimpleMovieLister">
    <constructor-arg ref="movieFinder"/>
</component>

The ref attribute is used to refer to another component by name.

If the component class has multiple constructors and/or a constructor has multiple arguments then it becomes necessary to disambiguate the constructor to be invoked and the order in which the arguments are passed to the constructor. The disambiguation is done based on the constructor signature (number and type of arguments).

Consider the following class:

```java
package x.y;

public class Foo {
    public Foo(Bar bar, Baz baz) {
        // ...
    }
}
```

Assuming that the Bar and Baz types only share java.lang.Object in common, the following configuration is valid:

```xml
<component name="fooOne" class="x.y.Foo">
    <constructor-arg ref="bar"/>
    <constructor-arg ref="baz"/>
</component>

<component name="fooTwo" class="x.y.Foo">
    <constructor-arg ref="baz"/>
    <constructor-arg ref="bar"/>
</component>
```

```xml
<component id="bar" class="Bar"/>
<component id="baz" class="Baz"/>
```
If instead the constructor of Foo had signature (Object, Object) then there would be no way to determine based on type which argument each of the constructor args should be bound to. In this case the declaration order of the constructor-arg elements is used determine the order in which arguments are bound.

Constructor arguments can also have their index specified explicitly by use of the index attribute. For example:

```
<component id="exampleComponent" class="examples.ExampleComponent">
  <constructor-arg index="0" value="7500000"/>
  <constructor-arg index="1" value="42"/>
</component>
```

The value attribute here is used to specify a string value that will be converted to the type required by the constructor argument (see the section 5.3.4.1.7 Type Conversion below). The declaration order of constructor-arg elements is not significant when the index attribute is used. The index attribute must be specified on every constructor-arg element if it is specified for any one of them. Indices are zero-based.

Consider a class that has multiple constructors with the same number of arguments, such as the class Multiple below:

```java
class Multiple {
    public Multiple(String s1, String s2) {...}
    public Multiple(int i1, int i2) {...}
}
```

An attempt to configure an instance of Multiple as shown below will fail, because it is not possible to determine which constructor should be invoked.

```
<component id="multiple" class="Multiple">
  <constructor-arg value="123"/>
  <constructor-arg value="456"/>
</component>
```

The type attribute can be used to disambiguate in such a case:

```
<component id="multiple" class="Multiple">
  <constructor-arg type="java.lang.String" value="123"/>
  <constructor-arg type="java.lang.String" value="456"/>
</component>
```

5.3.4.1.2 Setter Injection

Setter-based DI is realized by calling setter methods on your components after invoking a no-argument constructor or no-argument static factory method to instantiate your component. It is also possible to mix both constructor-based and setter-based injection for the same component. For example, mandatory dependencies could be specified via constructor injection, and optional dependencies via setter injection. Setter injection is available for properties of the component defined following JavaBeans conventions. The order in which properties are injected is undefined.

Here is an example:

```
<component id="exampleComponent" class="examples.ExampleComponent">
  <property name="componentOne" ref="anotherComponent"/>
  <property name="componentTwo" ref="yetAnotherComponent"/>
</component>
```
<property name="integerProperty" value="1"/>
</component>

<component id="anotherExampleComponent" class="examples.AnotherComponent"/>
<component id="yetAnotherComponent" class="examples.YetAnotherComponent"/>

public class ExampleComponent {
    private AnotherComponent compOne;
    private YetAnotherComponent compTwo;
    private int i;

    public void setComponentOne(AnotherComponent compOne) {
        this.compOne = compOne;
    }

    public void setComponentTwo(YetAnotherComponent compTwo) {
        this.compTwo = compTwo;
    }

    public void setIntegerProperty(int i) {
        this.i = i;
    }
}

As you can see, setters have been declared to match against the properties specified in the XML file, using
JavaBeans conventions.

5.3.4.1.3 Properties and configuration details
Component properties and constructor arguments can be defined as either references to other managed
components (collaborators), or values defined inline. A number of sub-element types are supported within the
<property/> and <constructor-arg/> elements for just this purpose.

The <value/> element specifies a property or constructor argument as a human-readable string representation.
The container converts these string values from a String to the actual type of the property or argument.

<component id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    <!-- results in a setDriverClassName(String) call -->
    <property name="driverClassName">
        <value>com.mysql.jdbc.Driver</value>
    </property>
    <property name="url">
        <value>jdbc:mysql://localhost:3306/mydb</value>
    </property>
    <property name="username">
        <value>root</value>
    </property>
    <property name="password">
        <value>masterkaoli</value>
    </property>
</component>

For cases where you need a more specific type conversion than the type of the property or argument to be
injected would imply, you can specify the fully-qualified name of the type that the string should be converted to
using the optional “type” attribute of the value element. For example, if property is declared with type “Object”, but you want the value injected to be of type BigDecimal you could specify:

```xml
<value type="java.math.BigDecimal">12345</value>
```

The `<property/>` and `<constructor-arg/>` elements also support the use of the ‘value’ attribute, which can lead to much more succinct configuration. When using the ‘value’ attribute, the above component definition reads like so:

```xml
<component id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource"
  destroy-method="close">
  <!-- results in a setDriverClassName(String) call -->
  <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
  <property name="url" value="jdbc:mysql://localhost:3306/mydb"/>
  <property name="username" value="root"/>
  <property name="password" value="masterkaoli"/>
</component>
```

The `<idref/>` element is simply an error-proof way to pass the id of another component in the container (to a `<constructor-arg/>` or `<property/>` element). The motivation for injecting a component id as opposed to a component itself is that the recipient may then use the ModuleContext API (defined in section 5.5) to lookup the component by name at a later date (after instantiation and configuration). Lookup by name can be useful when the component being looked up has a narrower scope than the component referencing it (scopes are discussed later in section 5.3.5) – the component instance returned may vary with each lookup under these circumstances.

```xml
<component id="theTargetComponent" class="..."/>
<component id="theClientComponent" class="...">
  <property name="targetName">
    <idref component="theTargetComponent"/>
  </property>
</component>
```

The above component definition snippet is exactly equivalent (at runtime) to the following snippet:

```xml
<component id="theTargetComponent" class="..."/>
<component id="theClientComponent" class="...">
  <property name="targetName" value="theTargetComponent"/>
</component>
```

The main reason the first form is preferable to the second is that using the idref tag allows the container to validate at deployment time that the referenced, named component actually exists. In the second variation, no validation is performed on the value that is passed to the ‘targetName’ property of the ‘client’ component. Any typo will only be discovered (with most likely fatal results) when the ‘client’ component is actually instantiated. If the ‘client’ component is a prototype component, this typo (and the resulting exception) may only be discovered long after the container is actually deployed.

The `<ref/>` element is the final element allowed inside a `<constructor-arg/>` or `<property/>` definition element. It is used to set the value of the specified property to be a reference to another component managed by the container (a collaborator). As mentioned in a previous section, the referred-to component is considered to be a dependency of the component whose property is being set, and will be initialized on demand as needed (if it is a singleton component it may have already been initialized by the container) before the property is set.
Specifying the target component by using the component attribute of the `<ref/>` tag is the most general form, and will allow creating a reference to any component in the same module context (whether or not in the same XML file). The value of the 'component' attribute may be the same as either the 'id' attribute of the target component, or one of the values in the 'name' attribute of the target component.

```xml
<ref component="someComponent"/>
```

### 5.3.4.1.4 Inner Components

A `<component/>` element inside the `<property/>` or `<constructor-arg/>` elements is used to define a so-called *inner component*. An inner component definition does not need to have any id or name defined, if these attributes are specified they will be ignored by the container.

```xml
<component id="outer" class="...">
  <!-- instead of using a reference to a target component, simply define the target component inline -->
  <property name="target">
    <component class="com.example.Person">
      <!-- this is the inner component -->
      <property name="name" value="Fiona Apple"/>
      <property name="age" value="25"/>
    </component>
  </property>
</component>
```

Note that in the specific case of inner components, the 'scope' attribute (see section 5.3.5) and any 'id' or 'name' attribute are effectively ignored. Inner components are *always* anonymous and always have *prototype* scope. Please also note that it is *not* possible to inject inner components into collaborating components other than the enclosing component. Inner components are are useful when:

- You have a component that should only be seen by its enclosing component, and
- You want to make it clear lexically that this is the case, and
- You want to avoid polluting the component namespace

### 5.3.4.1.5 Arrays

The `<array/>` element allows properties and arguments of array types to be defined and set. The `value-type` attribute specifies the type of the elements in the array. Nested inside the array element are the declarations for the array values themselves. An array value can be specified using any of the following elements:

- component
- reference
- service
- ref-set
- ref-list
- ref
- idref
- array
- list
- set
- map
- props
- value
- null

The reference, service, ref-set, and ref-list elements are introduced in section 5.4. Conversion will be attempted for each value to the type specified in the `array element's value-type attribute`. 
The following example shows the array element in use:

```xml
<component id="arrayExample" class="example.ArrayBean">
  <property name="myInts">
    <array value-type="int">
      <value>1</value>
      <value>2</value>
      <value>3</value>
      <value>5</value>
      <value>8</value>
    </array>
  </property>
  <property name="myStrings">
    <array value-type="java.lang.String">
      <value>one</value>
      <value>two</value>
      <value>three</value>
      <value>five</value>
      <value>eight</value>
    </array>
  </property>
</component>
```

### 5.3.4.1.6 Collections

The `list`, `set`, `map`, and `prop`s elements allow properties and arguments of the Java Collection type `List`, `Set`, `Map`, and `Properties`, respectively, to be defined and set.

```xml
<component id="moreComplexObject" class="example.ComplexObject">
  <!-- results in a setAdminEmails(java.util.Properties) call -->
  <property name="adminEmails">
    <props>
      <prop key="administrator">administrator@example.org</prop>
      <prop key="support">support@example.org</prop>
      <prop key="development">development@example.org</prop>
    </props>
  </property>
  <!-- results in a setSomeList(java.util.List) call -->
  <property name="someList">
    <list>
      <value>a list element followed by a reference</value>
      <ref component="myDataSource" />
    </list>
  </property>
  <!-- results in a setSomeMap(java.util.Map) call -->
  <property name="someMap">
    <map>
      <entry>
        <key>
          <value>an entry</value>
        </key>
        <value>just some string</value>
      </entry>
    </map>
  </property>
</component>
```
Note that the value of a map key or value, property value, constructor argument value, or a set or list value can be any of the following elements:

- component
- reference
- service
- ref-set
- ref-list
- ref
- idref
- array
- list
- set
- map
- props
- value

The value of an array, map, property, set, or list value can also be <null/>. The key and value of a prop element are always strings. The reference, service, ref-set, and ref-list elements are introduced in section 5.4.

If you are using Java 5 or Java 6, you will be aware that it is possible to have strongly typed collections (using generic types). For example, it is possible to declare a Collection type such that it can only contain String elements. If you are dependency injecting a strongly-typed Collection into a component, you can take advantage of type-conversion support such that the elements of your strongly-typed Collection instances will be converted to the appropriate type prior to being added to the Collection. Conversion is attempted for key values and for values specified using either the value attribute or element. Implementations may optionally support type conversion from other value types (e.g. map, list, set and so on).

Consider the following class which has a strongly typed Map property “accounts”:

```java
public class Foo {
    private Map<String, Float> accounts;

    public void setAccounts(Map<String, Float> accounts) {
        this.accounts = accounts;
    }
}
```
And the accompanying blueprint specification:

```
<components>
  <component id="foo" class="x.y.Foo">
    <property name="accounts">
      <map>
        <entry key="one" value="9.99"/>
        <entry key="two" value="2.75"/>
        <entry key="six" value="3.99"/>
      </map>
    </property>
  </component>
</components>
```

When the 'accounts' property of the 'foo' component is being prepared for injection the string values '9.99', '2.75', and '3.99' will be converted into objects of type Float. This feature is only support on Java 5 or higher runtime environments.

Regardless of JDK level, you can also specify an explicit conversion to be attempted for array, map, list, and set values, and for map keys, using the value-type and key-type attributes. These work in a similar manner to the type attribute previously described for the value element. If a `<map>` element has a `key-type` specified then this is assumed to be the default type for all keys within the map unless overridden on an individual key basis (by using the type attribute of a value element nested inside a key element). If an `<array>`, `<map>`, `<list>`, or `<set>` element has a `value-type` specified then this is assumed to be the default type for all values within the collection, unless overridden on an individual value basis (by using the type attribute of the value element).

Finally, a `<set>` or `<list>` can also be injected into a property of array type (for example, `int[]`), providing that each entry in the list or set can be successfully converted to the base type of the array. Type conversion is discussed next.

### 5.3.4.1.7 Type Conversion

String values in module context configuration files must be converted to the type expected by an injection target (method or constructor parameter, or field) before being injected. The module context container supports a number of type conversions by default, and provides an extension mechanism for configuring additional type converters.

The default type conversions supported by the module context container are:

- **String to any concrete type with a public String constructor**
  - If the target type has a public constructor that takes a single String argument, and no explicit converter is registered for that target type, then conversion is attempted by invoking the String constructor with the configuration value. This convention caters for many of the built-in JDK types such as `BigDecimal`, `BigInteger`, `Byte`, `Double`, `Float`, `Integer`, `Long`, `Short`, `File`, `URL`, `Date`, and so on.

- **String to primitive types**
  - With the exception of `boolean`, which permits additional values ("yes", "no", "true", "false", "on", "off"), the acceptable String formats for values are the same as defined for the String constructor of the associated wrapper type. Conversion to char supports the regular escape syntax for Java String literals.

- **String to Boolean (permissible values are "yes", "no", "true", "false", "on", "off")**

- **String to Character (the string should contain only one character, escape syntax is the same as for Java string literals)**

- **String to Locale. The String should follow the syntax:**

---

Copyright © The OSGi Alliance 2009  All Rights Reserved
Locale ::= language-code ("_" Country)+
Country ::= country-code ("_" variant-code)+
  o Where language-code, country-code and variant-code are two character codes as defined by the javadoc for the Locale class.

- String to Pattern (only support on JDK 1.5 or later)
- String to java.lang.Properties (String must follow the format described in the JavaDoc for Properties.load)

Implementations of the blueprint service may also define additional built-in converters.

Users may define their own converters by implementing the
org.osgi.service.blueprint.convert.Converter interface.

Interface Converter {
  public Class getTargetClass();
  public Object convert(Object source) throws Exception;
}

To register a converter it must be defined as a component and referenced from the type-converters element. Type converter elements are declared inside a components element before any component or other declarations. Type converter components may be declared directly inside the type-converters element using a component declaration, or referenced via a ref element. For example:

```xml
<type-converters>
  <!-- declare one or more type converters directly-->
  <component id="regionConverter" class="com.xyz.region.RegionConverter"/>

  <!--or simply reference components declared elsewhere -->
  <ref id="customDateConverter"/>
</type-converters>
```

All components declared within or referenced from a type-converters element must implement the Converter interface. Type converters are scoped to the module context and apply to all components defined for the context, regardless of the particular configuration file in which the converter or component may be defined.

Any converter declared in this way takes precedence over a built-in type converter when both are capable of performing the same conversion. If more than one explicitly declared type converter is capable of performing a given type conversion then:

- If the targetClass property (as returned by “getTargetClass”) of one and only one declared type converter exactly matches the property type of the property to be configured, then that converter will be used
- Otherwise it is unspecified which of the candidate type converters will be used.

For example, if a component to be configured has a property of type RegionCode, and the type EuropeanRegionCode extends RegionCode, then:

- Given a registered type converter with targetClass RegionCode and a registered type converter with targetClass EuropeanRegionCode, the RegionCode converter will be used.

---
3 See for example http://java.sun.com/j2se/1.4.2/docs/api/java/util/Properties.html#load(java.io.InputStream)
- Given two registered type converters with targetClass RegionCode one of the two will be used, but it is unspecified which one it will be.
- Given a registered type converter with targetClass EuropeanRegionCode, and another registered type converter with targetClass AsianRegionCode one of the two converters will be used, but it is unspecified which one it will be.

When declaring a component of type Converter, any configured properties of the converter component must rely only on the set of built-in type converters. This restriction also applies transitively to any components referenced by the converter component.

A bundle that exports a type implementing the Converter interface for use by other bundles should take care to specify the package of the targetClass type that the converter converts to in the uses clause for the package export. For example:

```
Export-Package: com.xyz.converters.region;uses:="com.xyz.region"
```

A bundle that references a type converter type defined locally does not need to export that type. When creating a module context, the extender bundle uses the class loader of the blueprint-defining bundle to load classes and resources for that context.

A module context contains an implicitly defined component with id “conversionService” of type org.osgi.service.blueprint.convert.ConversionService. Injecting a reference to the conversion service into a component provides access to the type conversions that the module context is able to perform.

### 5.3.4.1.8 Nulls
The `<null/>` element is used to handle null values. Empty arguments for properties and the like are treated as empty Strings. The following XML-based configuration metadata snippet results in the email property being set to the empty String value ("")

```
<component class="ExampleComponent">
  <property name="email"> <value/></value>
</property>
</component>
```

This is equivalent to the following Java code: exampleComponent.setEmail(""). The special `<null>` element may be used to indicate a null value. For example:

```
<component class="ExampleComponent">
  <property name="email"> <null/></null>
</property>
</component>
```

The above configuration is equivalent to the following Java code: exampleComponent.setEmail(null).

### 5.3.4.1.9 Configuration metadata shortcuts
The `<property/>`, `<constructor-arg/>`, and `<entry/>` elements all support a 'value' attribute which may be used instead of embedding a full `<value/>` element. Therefore, the following:

```
<property name="myProperty">
  <value>hello</value>
</property>

<constructor-arg>
  <value>hello</value>
</constructor-arg>
```
<entry key="myKey">
  <value>hello</value>
</entry>

are equivalent to:

<property name="myProperty" value="hello"/>
<constructor-arg value="hello"/>
<entry key="myKey" value="hello"/>

The <property/> and <constructor-arg/> elements support a similar shortcut 'ref' attribute which may be used instead of a full nested <ref/> element. Therefore, the following:

<property name="myProperty">
  <ref component="myComponent"/>
</property>

<constructor-arg>
  <ref component="myComponent"/>
</constructor-arg>

... are equivalent to:

<property name="myProperty" ref="myComponent"/>
<constructor-arg ref="myComponent"/>

Finally, the entry element allows a shortcut form to specify the key and/or value of the map, in the form of the 'key'/'key-ref' and 'value'/'value-ref' attributes. Therefore, the following:

<entry>
  <key>
    <ref component="myKeyComponent" />
  </key>
  <ref component="myValueComponent" />
</entry>

is equivalent to:

<entry key-ref="myKeyComponent" value-ref="myValueComponent"/>

5.3.4.1.10 Compound Property Names

Compound or nested property names are perfectly legal when setting component properties, as long as all components of the path except the final property name are not null. Consider the following component definition...

<component id="foo" class="foo.Bar">
  <property name="fred.bob.sammy" value="123" />
</component>
The foo component has a fred property which has a bob property, which has a sammy property, and that final sammy property is being set to the value 123. In order for this to work, the fred property of foo, and the bob property of fred must be non-null after the component is constructed, or a NullPointerException will be thrown.

### 5.3.4.2 Initialization guarantees

If a component A is directly or indirectly (for example via inclusion in a list or map) injected into a component B then in the absence of cyclic dependencies (see section 5.3.4.4), A is guaranteed to have been fully initialized before being injected.

### 5.3.4.3 Using depends-on

For most situations, the fact that a component is a dependency of another is expressed by the fact that one component is set as a property of another. This is typically accomplished with the `<ref/>` element. For the relatively infrequent situations where dependencies between components are less direct (for example, when a static initializer in a class needs to be triggered, such as database driver registration), the 'depends-on' attribute may be used to explicitly force one or more components to be initialized before the component using this element is initialized. Find below an example of using the 'depends-on' attribute to express a dependency on a single component.

```xml
<component id="compOne" class="ExampleComponent" depends-on="manager"/>
<component id="manager" class="ManagerComponent"/>
```

If you need to express a dependency on multiple components, you can supply a comma-delimited list of component names as the value of the 'depends-on' attribute.

```xml
<component id="compOne" class="ExampleComponent" depends-on="manager,accountDao">
    <property name="manager" ref="manager"/>
</component>
<component id="manager" class="ManagerComponent"/>
<component id="accountDao" class="x.y.jdbc.JdbcAccountDao"/>
```

The 'depends-on' attribute is used not only to specify an initialization time dependency, but also to specify the corresponding destroy time dependency (in the case of singleton components only). Dependent components that are defined in the 'depends-on' attribute will be destroyed after the relevant component itself being destroyed. This thus allows you to control shutdown order too.

### 5.3.4.4 Circular Dependencies

Dependency injection and the depends-on attribute create a dependency graph between components. An implementation of the blueprint service is not required to support cycles in the dependency graph. An implementation that does not support cycles should throw a ComponentDefinitionException (see section 5.5) if a cycle is detected. An implementation that provides some support for cyclic references may attempt to break cyclic dependencies by injecting a partially initialized (i.e. the object has been constructed, but all or some of the dependency injection setter methods may not yet have been invoked) component reference when possible. It is recommended that any such implementation issue a warning whenever such a situation occurs. When injecting a partially initialized component, the following guarantees must be made:

1. If a component declares an init method, it must be fully initialized before the init method is invoked
2. All components must be fully initialized before module context creation is complete (the CREATED event is posted, and the module context is published as a service).

If a cycle is detected that cannot be broken by such techniques, a ComponentDefinitionException must be thrown.
5.3.4.5 Lazily instantiated components

By default all singleton components in a module context will be pre-instantiated at startup. Pre-instantiation means that the module context will eagerly create and configure all of its singleton components as part of its initialization process. Generally this is a good thing, because it means that any errors in the configuration or in the surrounding environment will be discovered immediately (as opposed to possibly hours or even days down the line).

However, there are times when this behavior is not what is wanted. If you do not want a singleton component to be pre-instantiated you can selectively control this by marking a component definition as lazy-initialized. A lazily-initialized component is not created at startup and will instead be created when it is first requested.

Lazy loading is controlled by the 'lazy-init' attribute on the <component/> element; for example:

```xml
<component id="lazy" class="com.foo.ExpensiveToCreateComponent" lazy-init="true"/>
<component name="not.lazy" class="com.foo.AnotherComponent"/>
```

The component named 'lazy' will not be eagerly pre-instantiated when the module context is starting up, whereas the 'not.lazy' component will be eagerly pre-instantiated.

Even though a component definition may be marked up as being lazy-initialized, if the lazy-initialized component is the dependency of a singleton component that is not lazy-initialized, then when the module context is eagerly pre-instantiating the singleton, it will have to satisfy all of the singletons dependencies, one of which will be the lazy-initialized component! In this situation a component that you have explicitly configured as lazy-initialized will in fact be instantiated at startup; all that means is that the lazy-initialized component is being injected into a non-lazy-initialized singleton component elsewhere.

It is also possible to control lazy-initialization at the container level by using the 'default-lazy-init' attribute on the <components/> element; for example:

```xml
<components default-lazy-init="true">
  <!-- no components will be pre-instantiated... -->
</components>
```

The class referenced by the class attribute of a lazy-init component declaration is guaranteed not to be referenced in conjunction with that lazily initialized component until such time as the component is about to be instantiated.

5.3.5 Component Scopes

When you create a component definition what you are actually creating is a recipe for creating actual instances of the class defined by that component definition. The idea that a component definition is a recipe is important, because it means that, just like a class, you can potentially have many object instances created from a single recipe.

You can control not only the various dependencies and configuration values that are to be plugged into an object that is created from a particular component definition, but also the scope of the objects created from a particular component definition. This approach is very powerful and gives you the flexibility to choose the scope of the objects you create through configuration instead of having to 'bake in' the scope of an object at the Java class level. Components can be defined to be deployed in one of a number of scopes, specified using the scope attribute:

<table>
<thead>
<tr>
<th>Scope Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singleton</td>
<td>scopes a single component definition to a single object instance per module context</td>
</tr>
</tbody>
</table>
Prototype scopes a single component definition to any number of object instances

Bundle scopes a single component definition to a single object per requesting client bundle

When a component is a singleton, only one shared instance of the component will be managed, and all requests for components with an id or ids matching that component definition will result in that one specific component instance being returned by the container.

```xml
<component id="accountService" class="com.foo.DefaultAccountService" scope="singleton"/>
```

Singleton is the default scope if no scope is explicitly specified.

The non-singleton prototype scope of component deployment results in the creation of a new component instance every time the component is referenced for dependency injection, or is looked up using the `ModuleContext.getComponent` operation. Simply declaring a component with prototype scope does not on its own cause an instance to be created. As a rule of thumb, you should use the prototype scope for all components that are stateful, while the singleton scope should be used for stateless components.

In the following example `accountClientOne` and `accountClientTwo` will each be injected with their own unique instance of the `accountService` component:

```xml
<component id="accountService" class="com.foo.DefaultAccountService" scope="prototype"/>
<component id="accountClientOne" class="...">
    <property name="accountService" ref="accountService"/>
</component>
<component id="accountClientTwo" class="...">
    <property name="accountService" ref="accountService"/>
</component>
```

Note that naming a component with prototype scope in a depends-on attribute value will trigger creation of an “orphaned” instance of the prototype component and is not generally recommended.

See section 5.4.1 for a discussion of the bundle scope. Implementations of this RFC are free to define additional scope values beyond those defined here.

### 5.3.6 Lifecycle

Specifying an init-method for a component enables a component to perform initialization work once all the necessary properties on a component have been set by the container. Specifying a destroy-method enables a component to get a callback when the module context containing it is destroyed. Destroy-method callbacks are not supported for components with a prototype scope – it is the responsibility of the application to manage the lifecycle of prototype instances after they have been created. If the destroy-method attribute is used with a bean of prototype scope, it will be ignored.

```xml
<component id="exampleInitComponent"
    class="examples.ExampleComponent" init-method="init"/>
```

```java
public class ExampleComponent {
    public void init() {
        // do some initialization work
    }
}
```
<component id="exampleDestroyComponent" class="examples.ExampleComponent" destroy-method="cleanup"/>

public class ExampleComponent {
    public void cleanup() {
        // do some destruction work (like releasing pooled connections)
    }
}

The container can be configured to 'look' for named initialization and destroy callback method names on every component. This means that you, as an application developer, can simply write your application classes, use a convention of having an initialization callback called init() (or any other name of your choosing), and then (without having to configure each and every component with an 'init-method="init"' attribute) be safe in the knowledge that the container will call that method when the component is being created. To specify default init and destroy methods use the default-init-method and default-destroy-method attributes on the enclosing components element. For example:

<components default-init-method="onInit" default-destroy-method="onDestroy">
    <component id="someComponent" class="SomeClass">
        <!-- onInit() and onDestroy() methods will be called if implemented by SomeClass -->
    </component>
</components>

A component that specifies its own init-method or destroy-method attribute overrides any value specified in the corresponding default attribute of the components element. An init-method or destroy-method attribute with an empty value indicates that no init or destroy method (respectively) should be invoked.

If the type of a component does not have a method corresponding to a default-init-method or default-destroy-method attribute value (and the component definition does not override these values) then this is not treated as an error and the init or destroy method invocation is simply omitted. If the type of a component does not have a method corresponding to an init-method or destroy-method attribute value then this is an error and context creation will fail.

The method referenced by an init-method or default-init-method attribute must return void, and take no arguments. The method referenced by a destroy-method or default-destroy-method attribute must return void, and take no arguments.

The invocation of init methods follows component creation order (which follows the dependency graph). When an init-method is invoked on a component, any components that have been injected into the component will have already been instantiated and had their init-methods called. The invocation of destroy methods also follows component creation order (which follows the dependency graph). Thus when a component has its destroy method invoked, all of the components that component depends upon are still fully functional.

If any component code invoked during component instantiation and configuration (for example, constructors, setter methods, init methods) throws an exception then this is considered as fatal and context creation will fail. Any already constructed components will be destroyed (including invocation of destroy-methods if specified).
If a destroy-method implementation throws an exception the module context should catch the exception and recover to continue destroying any remaining components. If an implementation supports logging then the exception should be logged.

5.4 Interacting with the Service Registry

The osgi namespace provides elements that can be used to export managed components as OSGi services, to define references to services obtained via the registry.

5.4.1 Exporting a managed component to the Service Registry

The service element is used to define a component representing an exported OSGi service. The service element can be declared anywhere that a component element could be used. At a minimum you must specify the managed component to be exported, and the service interface that the service advertises.

For example, the declaration

```
<service ref="componentToPublish" interface="com.xyz.MessageService"/>
```

exports the component with name componentToPublish as a service in the OSGi service registry with interface com.xyz.MessageService. The published service will have a service property with the name osgi.service.blueprint.compname set to the component id of the target component being registered (componentToPublish in this case).

The component defined by the service element is of type org.osgi.framework.ServiceRegistration and is the ServiceRegistration object resulting from registering the exported component with the OSGi service registry. By giving this component an XML id you can inject a reference to the ServiceRegistration object into other components if needed. For example:

```
<service id="myServiceRegistration"
    ref="componentToPublish" interface="com.xyz.MessageService"/>
```

As an alternative to exporting a named component, the component to be exported to the service registry may be defined as an anonymous inner component of the service element.

```
<service interface="com.xyz.MessageService">
    <component class="SomeClass">
        ...
    </component>
</service>
```

In this case the blueprint service will not advertise an osgi.blueprint.service.compname service property for the registered service.

If the component to be exported implements the org.osgi.framework.ServiceFactory interface then the ServiceFactory contract is honored as per section 5.6 of the OSGi Service Platform Core Specification. As an alternative to implementing this OSGi API, this RFC introduces a component scope known as bundle scope. When a component with bundle scope is exported as an OSGi service then one instance of the component will be created for each unique client (service importer) bundle that obtains a reference to it through the OSGi service registry. Once created, a component with bundle scope has its lifecycle managed by the module context and will be disposed (and any associated destroy-method invoked) when either the containing module context is disposed, or the reference count from the service-importing bundle drops to zero (e.g. that bundle is stopped). If the service importing bundle subsequently references the component again after previously releasing all references, a new instance will be created.

To declare a component with bundle scope use the scope attribute of the component element:
A component with bundle scope that is not exported as a service simply behaves as a component with scope singleton.

### 5.4.1.1 Controlling the set of advertised service interfaces for an exported service

The OSGi Service Platform Core Specification defines the term service interface to represent the specification of a service's public methods. Typically this will be a Java interface, but the specification also supports registering service objects under a class name, so the phrase service interface can be interpreted as referring to either an interface or a class.

There are several options for specifying the service interface(s) under which the exported service is registered. The simplest mechanism, shown above, is to use the interface attribute to specify a fully-qualified interface name. To register a service under multiple interfaces the nested `<interfaces>` element can be used in place of the `interface` attribute.

```xml
<service ref="componentToBeExported">
  <interfaces>
    <value>com.xyz.MessageService</value>
    <value>com.xyz.MarkerInterface</value>
  </interfaces>
</service>
```

The `interface` attribute must not be used in conjunction with the `<interfaces>` element.

Using the `auto-export` attribute you can avoid the need to explicitly declare the service interfaces at all by analyzing the object class hierarchy and its interfaces.

The `auto-export` attribute can have one of four values:

- **disabled**: the default value; no auto-detected of service interfaces is undertaken and the interface attribute or `<interfaces>` element must be used instead.
- **interfaces**: the service will be registered using all of the Java interface types implemented by the component to be exported.
- **class-hierarchy**: the service will be registered using the exported component's implementation type and super-types (up to but not including java.lang.Object).
- **all-classes**: the service will be registered using the exported component's implementation type and super-types (up to but not including java.lang.Object) plus all interfaces implemented by the component.

For example, to automatically register a component under all of the interfaces that it supports you would declare:

```xml
<service ref="componentToBeExported" auto-export="interfaces"/>
```

Given the interface hierarchy:

```java
public interface SuperInterface {}
public interface SubInterface extends SuperInterface {}
```

then a service registered as supporting the `SubInterface` interface is not considered a match in OSGi when a lookup is done for services supporting the `SuperInterface` interface. For this reason it is a best practice to
export all interfaces supported by the service being registered explicitly, using either the `interfaces` element or `auto-export="interfaces"`.

5.4.1.2 Controlling the set of advertised properties for an exported service

As previously described, an exported service is always registered with the service property `osgi.service.blueprint.compname` set to the name of the component being exported. Additional service properties can be specified using the nested `service-properties` element. The `service-properties` element contains key-value pairs to be included in the advertised properties of the service. The key must be a string value, and the value must be a type recognized by OSGi Filters. See section 5.5 of the OSGi Service Platform Core Specification for details of how property values are matched against filter expressions.

The `service-properties` element must contain at least one nested `entry` element. For example:

```
<service ref="componentToBeExported" interface="com.xyz.MyServiceInterface">
  <service-properties>
    <entry key="myOtherKey" value="aStringValue"/>
    <entry key="aThirdKey" value-ref="componentToExposeAsProperty"/>
  </service-properties>
</service>
```

See section 5.5 for details on how to register service properties sourced from the Configuration Admin service.

When registering a property for a service, you must explicitly specify the value type unless you wish the value to be registered as a string. The OSGi specification (section 3.2.6) allows the following values for service properties:

- A primitive (int, long, float, double, byte, short, char, boolean). A value with one of these types can be specified using e.g.
  ```
  <entry key="someKey">
    <value type="int">123</value>
  </entry>
  ```

- One of the scalar types String, Integer, Long, Float, Double, Byte, Short, Character, Boolean. A value with one of these types can be specified using e.g.
  ```
  <entry key="someKey">
    <value type="java.lang.Integer">123</value>
  </entry>
  ```
  (Remember that no special treatment is needed for String values)

- An array of either the allowable primitive or scalar types. A value with one of these types can be specified using e.g.
  ```
  <entry key="someKey">
    <array value-type="java.lang.String">
      <value>first</value>
      <value>second</value>
    </array>
  </entry>
  ```
• A Collection of scalar types. A value with one of these types can be specified using e.g.
  
  <entry key="someKey">
    <set>
      <value>First</value>
      <value>Second</value>
    </set>
  </entry>

5.4.1.3 The depends-on attribute

The optional depends-on attribute can be used to provide a comma-delimited list of component names for components that must be instantiated and configured before the service is published to the registry. The depends-on attribute is used to capture indirect dependencies required for operation of the service that cannot be explicitly determined through the configuration of the component to be exposed as a service. Dependencies listed in the depends-on clause only gate service registration, they do not propagate to the creation of the component referenced by the service element using the ref attribute.

5.4.1.4 The ranking attribute

When registering a service with the service registry, you may optionally specify a service ranking (see section 5.2.5 of the OSGi Service Platform Core Specification). When a bundle looks up a service in the service registry, given two or more matching services the one with the highest ranking will be returned. The default ranking value is zero. To explicitly specify a ranking value for the registered service, use the optional ranking attribute.

<service ref="componentToBeExported" interface="com.xyz.MyServiceInterface"
  ranking="9"/>

5.4.1.5 Registration Listener

The service defined by a service element is registered with the OSGi service registry when the module context is first created. It will be unregistered automatically when the bundle is stopped and the module context is disposed. Services are also unregistered and re-registered if a mandatory dependency of the service is unsatisfied or becomes satisfied again (see section 5.4.2). The ServiceRegistration object representing the service component is a proxy that always delegates to the current OSGi service platform ServiceRegistration for the service. If an operation is invoked on the ServiceRegistration proxy at a time when there is no underlying ServiceRegistration (because the service has been unregistered, but not yet re-registered) then an IllegalStateException will be thrown.

If you need to take some action when a service is registered or unregistered then you can define a listener component using the nested registration-listener element.

The declaration of a registration listener must use either the ref attribute to refer to a top-level component definition, or declare an anonymous listener component inline. For example:

<service ref="componentToBeExported" interface="SomeInterface">
  <registration-listener ref="myListener"                   (1)
    registration-method="serviceRegistered"                (2)
    unregistration-method="serviceUnregistered"/>          (2)
  <registration-listener
    registration-method="onRegister"
unregistration-method="onUnregister"/>
  <component class="SomeListenerClass"/>
</registration-listener>
</service>

(1) Listener declaration referring to a top-level component.

(2) The registration and unregistration methods to be invoked on the component referenced in (1).

(3) The registration and unregistration methods to be invoked on the component defined in (4)

(4) Listener component declared anonymously in-line.

The required registration-method and unregistration-method attributes specify the names of the methods defined on the listener component that are to be invoked during registration and unregistration. Registration and unregistration callback methods must have a signature matching with the following format:

```
public void anyMethodName(ServiceType serviceInstance, Map serviceProperties);
```

where ServiceType can be any type compatible with the exported service interface of the service.

The register callback is invoked when the service is initially registered at startup, and whenever it is subsequently re-registered. The unregister callback is invoked during the service unregistration process, no matter the cause (such as the owning bundle stopping). The ServiceRegistration proxy representing the service component is guaranteed to be backed by a valid ServiceRegistration object during register and unregister callbacks.

The registration/unregistration methods are only invoked when a service of a type compatible with the declared ServiceType is registered/unregistered.

The components referred to by registration-listener elements (either via the ref attribute or by direct inline declaration of a component) are treated as dependencies of the enclosing service component for the purposes of lifecycle management, dependency injection, and cycle detection. Given a service component S, with a listener L, it must however be possible to inject a reference to S into L. For example:

```
<service id="S" ref="SomeComponent" interface="I">
  <registration-listener
    registration-method="onRegister"
    unregistration-method="onUnregister"
    ref="L"/>
</registration-listener>
</service>

<component id="L" class="C">
  <property name="serviceRegistration" ref="S"/>
</component>
```
5.4.1.6 Lazy Activation

When a bundle specifies lazy activation in its Bundle-ActivationPolicy manifest header, and that bundle is started with the “START_ACTIVATION_POLICY” flag set, then the bundle will wait in the STARTING state until the first class load from the bundle occurs. For a bundle with a module context blueprint that defines one or more services, this means that without special treatment, none of these services would be visible until a class load occurs. An important use case is to enable the first dereference of a service reference to trigger creation of the module context.

For a bundle that specifies the lazy activation policy and is started accordingly, the following steps are taken:

- The OSGi Service Platform creates a BundleContext object for the bundle
- The bundle state is moved to the STARTING state
- The LAZY_ACTIVATION event is fired
- A synchronous Blueprint Service event listener receives the LAZY_ACTIVATION event
  - If an EventAdmin service is present, an org/osgi/service/blueprint/context/CREATING event is published
  - The configuration metadata for the bundle is parsed, and any service elements are detected
  - The Blueprint Service waits for any mandatory service references of the context to be satisfied, according to the rules discussed in section 5.2. Any timeouts are relative to the receipt of the LAZY_ACTIVATION event (not the STARTING or STARTED events). Once the services are satisfied:
    - For any service that can be lazily registered, a lazy ServiceFactory is registered in the service registry to represent that service
      - A service can be lazily registered if the interface or interfaces under which the service is to be registered are explicitly specified and are Java interface types, and there are no registration listeners defined.
      - The lazy service advertises the same interfaces and properties as the ‘real’ service that will be created when the module context is created.
    - No components from the module context are instantiated during this process
    - The synchronous event listener returns

From this point, there are two triggers that can cause the module context to be created: a class load from the bundle, or a dereferencing of the ServiceReference (invocation of getService on the registered ServiceFactory for the service) for a lazy service.

In the first case:

- The system waits for a class load from the bundle to occur
- The normal STARTING event is fired by the framework
- The bundle is activated by the framework
- The STARTED event is fired and the framework moves the bundle to the ACTIVE state
- Asynchronous creation of the module context begins on receipt of the STARTED event
  - Any lazy ServiceFactory objects registered during LAZY_ACTIVATION are updated to return the true services defined in the module blueprint.
In the second case:

- The ServiceFactory method `getServiceImpl` is invoked on a lazy service (because a client has passed the `ServiceReference` obtained from the registry to the `BundleContext.getService` operation).
- The `Bundle.start(START_TRANSIENT)` operation is invoked to force the bundle to start immediately
- The normal `STARTING` event is fired by the framework
- The bundle is activated by the framework
- The `STARTED` event is fired and the framework moves the bundle to the `ACTIVE` state
- *Synchronous* creation of the module context begins

  - Any lazy `ServiceFactory` objects registered during `LAZY_ACTIVATION` are updated to return the true services defined in the module blueprint.
  - Module context creation completes

- The `getService` operation returns the true service from the newly created module context

If module context creation (however triggered) fails for any reason, then any registered placeholder services for that context must be unregistered.

Note that the potential for race conditions exists under concurrent class loading and/or service access. An implementation of the Blueprint Service must ensure that only one module context is instantiated under such conditions.

### 5.4.2 Defining References to OSGi Services

The Blueprint Service supports the declaration of components that represent services accessed via the OSGi Service Registry. In this manner references to OSGi services can be injected into bundle components. The service lookup is made using the service interface type that the service is required to support, plus an optional filter expression that matches against the service properties published in the registry.

For some use cases, a single matching service that meets the application requirements is all that is needed. The `reference` element defines a reference to a single service that meets the required specification. In other scenarios, especially when using the OSGi whiteboard pattern, references to all available matching services are required. The Blueprint Service supports the management of this set of references as a List or Set. References to services can be declared anywhere that a component element could be declared.

#### 5.4.2.1 Referencing an individual service

The `reference` element is used to define a reference to a service in the service registry.

Since there can be multiple services matching a given description, the service returned is the service that would be returned by a call to `BundleContext.getServiceReference`. This means that the service with the highest ranking will be returned, or if there is a tie in ranking, the service with the lowest service id (the service registered first with the framework) is returned (please see Section 5 from the OSGi specification for more information on the service selection algorithm).

**Interface attribute and interfaces element**

The `interface` attribute identifies the service interface that a matching service must implement. For example, the following declaration creates a reference component called `messageService`, which is backed by the service returned from the service registry when querying it for a service offering the `MessageService` interface.
<reference id="messageService" interface="com.xyz.MessageService"/>

Just as with the service element, when specifying multiple interfaces, use the nested interfaces element instead of the interface attribute:

`<reference id="importedOsgiService">
  <interfaces>
    <value>com.xyz.MessageService</value>
    <value>com.xyz.MarkerInterface</value>
  </interfaces>
</reference>`

It is illegal to use both the interface attribute and the interfaces element at the same time.

The component defined by the reference element implements all of the advertised service interfaces of the service that are visible to the bundle. Implementations of this RFC may choose to document a limitation that class-based (as opposed to interface-based) service interfaces that include final methods, or classes that are final themselves, are not supported.

Filter attribute
The optional filter attribute can be used to specify an OSGi filter expression and constrains the service registry lookup to only those services that match the given filter.

For example:

`<reference id="asyncMessageService" interface="com.xyz.MessageService" filter="(asynchronous-delivery=true)"/>`

will match only OSGi services that advertise the MessageService interface and have the property named asynchronous-delivery set to value 'true'.

Component name attribute
The component-name attribute is a convenient short-cut for specifying a filter expression that matches on the component name property automatically set when exporting a component using the service element.

For example:

`<reference id="messageService" interface="com.xyz.MessageService" component-name="defaultMessageService"/>`

will match only OSGi services that advertise the MessageService interface and have the property named osgi.service.blueprint.compname set to value defaultMessageService.

If both a filter attribute value and a component-name attribute value are specified, then matching services must satisfy the constraints of both.

Availability attribute
The availability attribute is used to specify whether or not a matching service is required at all times. An availability value of mandatory (the default) indicates that a matching service must always be available. An availability value of optional indicates that a matching service is not required at all times. A reference with
mandatory availability is also known as a mandatory service reference and, by default, module context creation is deferred until the reference is satisfied.

Note: It is an error to declare a mandatory reference to a service that is also exported by the same bundle, this behavior can cause module context creation to fail through either deadlock or timeout.

Obtaining a ServiceReference object

If the property into which a reference component is to be injected has type ServiceReference (instead of the service interface supported by the reference), then an OSGi ServiceReference for the service, as provided by the OSGi Service Platform in which the application is running, will be injected in place of the service itself.

The injected service reference refers to the service instance satisfying the reference at the time the reference is injected. The ServiceReference object will not be updated if the backing service later changes. If there is no matching service instance at the time of injection (for example, the reference is to an optional service), then 'null' will be injected.

For example, given the following Java class declaration and component declarations:

```java
public class ComponentWithServiceReference {
    private ServiceReference serviceReference;
    private SomeService service;
    // getters/setters ommitted
}
```

```xml
<reference id="service" interface="com.xyz.SomeService"/>
<component id="someComponent" class="ComponentWithServiceReference">
    <property name="serviceReference" ref="service"/>
    <property name="service" ref="service"/>
</component>
```

Then

1. The ServiceReference object for the service obtained via the reference element will be injected into the serviceReference property.
2. An object representing the service itself will be injected into the service property

5.4.2.2 Referencing a collection of services

Sometimes an application needs access not simply to any service meeting some criteria, but to all services meeting some criteria. The matching services may be held in a List or Set (optionally sorted).

The difference between using a List and a Set to manage the collection is one of equality. Two or more services published in the registry (and with distinct service ids) may be "equal" to each other, depending on the implementation of equals used by the service implementations. Only one such service will be present in a set, whereas all services returned from the registry will be present in a list. The ref-set and ref-list schema elements are used to define collections of services with set or list semantics respectively.

These elements support all of the attributes and nested elements defined for the reference element. An availability value of optional indicates that it is permissible for there to be no matching services. An availability value of mandatory indicates that at least one matching service is required at all times. Such a reference is considered a mandatory reference and any exported services from the same bundle (service defined
components) that depend on a mandatory reference will automatically be unregistered when the reference becomes unsatisfied, and re-registered when the reference becomes satisfied again.

The component defined by a ref-list element is of type java.util.List. The component defined by a ref-set element is of type java.util.Set.

The following example defines a component of type List that will contain all registered services supporting the EventListener interface:

```xml
<ref-list id="myEventListeners" interface="com.xyz.EventListener"/>
```

The members of the collection defined by the component are managed dynamically. As matching services are registered and unregistered in the service registry, the collection membership will be kept up to date. Each member of the collection supports the service interfaces that the corresponding service was registered with and that are visible to the bundle.

Sorted collections are also supported. To specify that a collection should be sorted, the optional ordering-basis attribute of ref-list and ref-set can be specified. The ordering-basis attribute can take one of two values: service or service-reference. If service is specified then sorting is based on the service instances in the collection. If service-reference is specified then sorting is based on the corresponding ServiceReference objects for those services. A comparator to use for the sorting can be specified using either the comparator-ref attribute, or the nested comparator element. The comparator-ref attribute is used to refer to a named component implementing java.util.Comparator. The comparator will be passed either the service objects or the ServiceReference objects, according to the value of the ordering-basis attribute. The comparator element can be used to define an inline component. For example:

```xml
<ref-set id="myServices" interface="com.xyz.MyService" ordering-basis="service"
        comparator-ref="someComparator"/>
```

```xml
<ref-list id="myOtherServices" interface="com.xyz.OtherService" ordering-basis="service-reference">
  <comparator>
    <component class="MyOtherServiceComparator"/>
  </comparator>
</ref-list>
```

If no comparator is specified (neither the comparator element nor comparator-ref attribute are used) then sorting occurs based on natural ordering. If a comparator is specified, but no ordering-basis, then the comparator will be passed the service instances.

**Obtaining Service Reference Objects**

If the property into which a reference set or list is to be injected is of type Collection<ServiceReference> or a subtype thereof (e.g. List<ServiceReference>, Set<ServiceReference>), then the injection collection will contain the ServiceReference objects for the matching services rather than the service objects themselves.

To support JDK 1.4 and below where generic types are not available, the optional member-type attribute of ref-set and ref-list allows you to specify explicitly the desired type of collection member. Permissible values are service-instance (the default behaviour) and service-reference (in which case the collection will contain ServiceReference objects).
5.4.3 Dealing with service dynamics

The component defined by a reference element is unchanged throughout the lifetime of the module context (the object reference remains constant). However, the OSGi service that backs the reference may come and go at any time. For a mandatory service reference, creation of the module context will block until a matching service is available. For an optional service reference (optional availability), the reference component will be created immediately, regardless of whether or not there is currently a matching service.

When the service backing a reference component goes away, an attempt is made to replace the backing service with another service matching the reference criteria. An application may be notified of a change in backing service by registering a listener (see section 5.4.3.2). If no matching service is available, then the reference is said to be unsatisfied. An unsatisfied mandatory service causes any exported service (service component) that depends on it to be unregistered from the service registry until such time as the reference is satisfied again.

When an operation is invoked on an unsatisfied reference component (either optional or mandatory), the invocation blocks until either the reference becomes satisfied or a timeout expires (whichever comes first). If an EventAdmin service is present then an org.osgi.service.blueprint/context/WAITING event is broadcast as described in section 5.2.3. The default timeout for service invocations is 5 minutes. The optional timeout attribute of the reference element enables an alternate timeout value (in milliseconds) to be specified. If no matching service becomes available within the timeout period, an unchecked ServiceUnavailableException is thrown. Specifying a timeout value of zero or less means that there will be no timeout period, and an unsatisfied service invocation will fail immediately.

Consider the following simple example:

```
<reference id="service" interface="com.xyz.SomeService" availability="optional" />
<component id="someComponent" class="SomeComponent">
  <property name="service" ref="service"/>
</component>
```

When the module context is created, and no service supporting the SomeService interface type is available in the registry, then a component instance is instantiated for the service component, but no backing service is set for it. The someComponent component is then instantiated and injected with a reference to the service component. At some point later, a service is published in the service registry that supports the SomeService interface. This service is transparently set as the backing service behind the service component. If an invocation is made on the service component when it has no backing service available, the timeout rules described above would then come into play.

Note that a module context with mandatory service references will by default wait for all mandatory service references to be satisfied before the context is initially created, but once the context has been created, a mandatory service reference that becomes unsatisfied does not cause the whole context to be disposed. The rationale is that having all mandatory services satisfied should be the normal case, and hence bringing up a context in a partially satisfied state is undesirable. However, once the context is created, temporary absences of mandatory services are tolerated to allow for administration operations and continuous operation of as much of the system as possible. For a reference collection with mandatory availability, the reference is considered satisfied if there is at least one matching service in the collection.

The timeout attribute is not supported by the ref-set and ref-list elements.

While a reference component will try to find a replacement if the backing service is unregistered, a reference collection-based component will simply remove the service from the collection. The recommend way of traversing a collection is by using an Iterator. During iteration, all Iterators held by the user will be transparently updated so it is possible to safely traverse the collection while it is being modified. Moreover, the Iterators will reflect all the changes made to the collection, even if they occurred after the Iterators were created (that is, during the iteration). Consider a case where a collection shrinks significantly (for example a large number of OSGi services are shutdown) right after an iteration started. To avoid dealing with the resulting 'dead' service
references, iterators do not take collection snapshots but instead are updated on each service event so they reflect the latest collection state, no matter how fast or slow the iteration is.

It is important to note that a service update will only influence Iterator operations that are executed after the event occurred. Services already returned by the iterator will not be updated even if the backing service has been unregistered. If an operation is invoked on such a service that has been unregistered, a ServiceUnavailableException will be thrown.

The Iterator contract is guaranteed: the next() method always obeys the result of the previous hasNext() invocation. Within this contract, an implementation is free to add additional matching elements into the collection during iteration, and to remove as yet unseen elements from the collection during iteration. A client may therefore see the return value of repeated calls to hasNext() change over time: for example after returning false a new member may be added to the collection causing a subsequent invocation of hasNext() to return true. The next() method always obeys the result of the previous hasNext() invocation, so if hasNext() returns true there is guaranteed to be an available object on a call to next().

Any elements added to the collection during iteration over a sorted collection will only be visible if the iterator has not already passed their sort point.

Collections of ServiceReferences are managed in the same way as collections of the service objects themselves (i.e. ServiceReference objects may be added and removed dynamically so long as the Iterator contract is honored).

5.4.3.1 Mandatory dependencies

An exported service may depend, either directly or indirectly, on other services in order to perform its function. If one of these services is considered a mandatory dependency (has 'mandatory' availability) and the dependency can no longer be satisfied (because the backing service has gone away and there is no suitable replacement available) then the exported service that depends on it will be automatically unregistered from the service registry - meaning that it is no longer available to clients. If the mandatory dependency becomes satisfied once more (by registration of a suitable service), then the exported service will be re-registered in the service registry.

This automatic unregistering and re-registering of exported services based on the availability of mandatory dependencies only takes into account declarative dependencies. If exported service S depends on component A, which in turn depends on mandatory imported service M, and these dependencies are explicit in the module configuration file as per the example below, then when M becomes unsatisfied S will be unregistered. When M becomes satisfied again, S will be re-registered.

```
<service id="S" ref="A" interface="SomeInterface"/>

<component id="A" class="SomeImplementation">
  <property name="helperService" ref="M"/>
</component>

<reference id="M" interface="HelperService"
  availability="mandatory"/>
```

If however the dependency from A on M is not established through configuration as shown above, but instead at runtime through for example storing a reference to M in a field belonging to A without any involvement from the container, then this dependency is not tracked.

Automatic service registration and unregistration also applies to lazily registered services that have been registered on behalf of a bundle with a LAZY_ACTIVATION policy.
5.4.3.2 Service Listeners

Applications that need to be aware of when a service backing a reference component is bound and unbound, or when a member is added to or removed from a collection, can register one or more listeners using the nested listener element. The listener element refers to a component (either by name using the ref attribute or nested ref element, or by defining one inline) that will receive bind and unbind notifications. The bind-method and unbind-method attributes indicate the operations to be invoked on the listener component during a bind or unbind event respectively.

For example:

```xml
<reference id="someService" interface="com.xyz.MessageService">
  <listener bind-method="onBind" unbind-method="onUnbind">
    <component class="MyCustomListener"/>
  </listener>
</reference>
```

The signature of a custom bind or unbind method must be one of:

```java
public void anyMethodName(ServiceType service, Map properties);
public void anyMethodName(ServiceReference ref);
```

where ServiceType can be any type. The bind and unbind callbacks are invoked only if the service instance is assignable to a reference of type ServiceType. The properties parameter contains the set of properties that the service was registered with.

If the method signature has a single argument of type ServiceReference then the ServiceReference of the service will be passed to the callback in place of the service object itself.

When the listener is used with a reference declaration:

- A bind callback is invoked when the reference is initially bound to a backing service, and whenever the backing service is replaced by a new backing service.
- An unbind callback is only invoked when the current backing service is unregistered, and no replacement service is immediately available (i.e., the reference becomes unsatisfied).

When the listener is used with a collection declaration (set or list):

- A bind callback is invoked when a new service is added to the collection.
- An unbind callback is invoked when a service is unregistered and is removed from the collection.

Bind and unbind callbacks are made synchronously as part of processing an OSGi serviceChanged event for the backing OSGi service, and are invoked on the OSGi thread that delivers the corresponding OSGi ServiceEvent. In particular, the service or ServiceReference passed to a bind or unbind callback is valid for the lifetime of the callback. For collection based callbacks, the service 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 listener component defines multiple (overridden) methods with a name specified in a bind or unbind method attribute, then every method with a compatible signature (including both ServiceType and ServiceReference forms) is invoked on the corresponding event. The order of invocation of the matching methods is undefined.

The components referred to by listener elements (either via the ref attribute or by direct inline declaration of a component) are treated as dependencies of the enclosing service reference or collection component for the purposes of lifecycle management, dependency injection, and cycle detection. Given a reference component R, with a listener L, it must however be possible to inject a reference to R into L. For example:
<reference id="R" interface="I">
  <listener bind-method="onBind"
    unbind-method="onUnbind"
    ref="L"/>
  
  </listener>
</reference>

<component id="L" class="C">
  <property name="observedService" ref="R"/>
</component>

### 5.4.3.3 Module-wide defaults for service references

The `components` element supports the setting of `default-availability` and `default-timeout` attribute values that then serve as the defaults for the `availability` attribute of the `reference`, `ref-set`, and `ref-list` elements, and the `timeout` attribute of the `reference` element respectively, when no value is specified.

### 5.5 Module Context API

### 5.6 Namespace Extension Mechanism

Third parties may contribute additional namespaces containing elements and attributes used to configure the components for a module context. These additional namespaces are referenced in the standard XML manner using the `xmlns` attribute of the top level element. The following configuration file references the osgi, osgix, and aop namespaces:

```xml
<components xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:osgix="http://www.osgi.org/xmlns/blueprint-compendium/v1.0.0"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="
    http://www.osgi.org/xmlns/blueprint/v1.0.0
    http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
    http://www.osgi.org/xmlns/blueprint-compendium/v1.0.0
    http://www.osgi.org/xmlns/blueprint-compendium/v1.0.0/blueprint-compendium.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd"
>
  ...
</components>
```

In order to be able to interpret elements and attributes declared in third party namespaces, a namespace handler must be registered that the container can delegate to. Namespace handlers are registered in the OSGi service registry under the `org.osgi.service.blueprint.namespace.NamespaceHandler` interface. A registered namespace handler must also advertise the service property `osgi.service.blueprint.namespace`. The value of this property is an array of URIs for the schema that the namespace can handle (for example, [http://www.springframework.org/schema/aop](http://www.springframework.org/schema/aop)). When defining a namespace extension and schema the following conventions are recommended (but not required):

- A schema URI with a final path segment of the format “v[version]” – for example, [http://www.foobar.org/schema/foo/v1.0.0](http://www.foobar.org/schema/foo/v1.0.0) references that version of the schema (1.0.0 in this case)
• A schema URI without such a final path segment references the latest version of the schema
• A namespace handler should publish in its osgi.service.blueprint.namespace property the URIs of each version of the schema it can handle, together with the unqualified URI if the handler can also handle the most recent version.

During the processing of a configuration file the container must determine the set of schemas used to define the elements in the configuration file. The resulting set of schema uris (for the above example this would be http://www.osgi.org/schema/service/blueprint, http://www.osgi.org/schema/service/blueprint-compendium, and http://www.springframework.org/schema/aop) determine the namespace handlers that must be present in order to process the file. For each schema uri the container creates a mandatory service reference for a service implementing the NamespaceHandler interface and with a matching osgi.service.blueprint.namespace property value. Creation of the context waits for these service references to be satisfied in the same way as it would for any service reference explicitly declared via a service element – including generation of the wait events if an EventAsadmin service is available and the services are not immediately available.

The NamespaceHandler interface is defined as follows:

interface NamespaceHandler {
    /**
     * The URL where the xsd file for the schema may be found. Typically used to return a URL to a
     * bundle resource entry so as to avoid needing to lookup schemas remotely.
     * If null is returned then the schema location will be determined from the xsi:schemaLocation
     * attribute
     * value.
     */
    URL getSchemaLocation();

    /**
     * Called when a top-level (i.e. non-nested) element from the namespace is encountered.
     * Implementers may register component definitions themselves, and/or return a component definition
     * to be registered.
     */
    ComponentMetadata parse(org.w3c.dom.Element element, ParserContext context);

    /**
     * Called when an attribute or directly nested element is encountered. Implementors should parse the
     * supplied Node and decorated the provided component, returning the decorated component.
     */
    ComponentMetadata decorate(org.w3c.dom.Node node, ComponentMetadata component,
                              ParserContext context);
}

The ParserContext passed to the parse and decorate methods provides access to the ComponentMetadata of the enclosing component (if any) and to the ComponentDefinitionRegistry.

The parse callback is invoked when an element from the namespace is encountered that is not directly nested inside of a component, service, reference, ref-list, or ref-set element. Most commonly it is invoked
when a top-level namespace element is encountered. Very often a custom namespace element serves as a
convenient shortcut for creating a component definition. In this case where one namespace element corresponds
to one component definition, the simplest implementation is often to return the new component definition as the
return value of the parse callback. In more complex cases, a single namespace element may map to a collection
of component definitions, or conditionally create components. In such circumstances the context passed to the
parse method can be used to discover existing component definitions and to explicitly register new ones.

The `decorate` callback is invoked when an attribute from the handled namespace is encountered in an element
that is not from the handled namespace. Decorate typically modifies the component definition for the component
in which the attribute was encountered, and returns the modified definition. The `decorate` callback is also
invoked when an element from the handled namespace is encountered nestly directly inside a `component`,
`service`, `reference`, `ref-list`, or `ref-set` element.

Consider the following configuration snippet involving a hypothetical "cache" namespace:

```
<cache:lru-cache id="myCache"/>

<component id="fooService" class="FooServiceImpl"
   cache:cache-return-values="true">
   <cache:exclude>
      <cache:operation name="getVolatile"/>
   </cache:exclude>

   <property name="myProp" value="12"/>
</component>

<component id="barService" class="BarServiceImpl">
   <property name="localCache">
      <cache:lru-cache/>
   </property>
</component>
```

Given a namespace handler registered to handle the cache namespace, then at:

1. The `parse` method will be invoked, and will typically return a new component definition defining a cache
   component.

2. The `decorate` method will be invoked passing in the `cache-return-values` attribute Node, and the
   `ComponentMetadata` for the `fooService` component.

3. The `decorate` method will be invoked passing in the `exclude` element Node, and the
   `ComponentMetadata` for the `fooService` component.

4. There is no callback for this line, it is the responsibility of the `decorate` method at (3) to process the
   contents of the `exclude` element.

5. The `parse` method will be invoked and would typically return a new component definition defining a
   cache component. From the `ParserContext` passed to the method, `getEnclosingComponent` will return the
   component definition for the `barService` component.
5.6.1 Date Namespace Example

The following example illustrates how the Blueprint Service could be extended to process elements from a "date" namespace. In our example, the date namespace contains only one element, dateformat. The namespace handler will enable component configuration files to include declarations such as:

```xml
<date:dateformat id="dateFormat"
pattern="yyyy-MM-dd HH:mm"
lenient="true"></date:dateformat>
```

and such declarations will in effect be equivalent to the following component definition:

```xml
<component id="dateFormat" class="java.text.SimpleDateFormat">
  <constructor-arg value="yyyy-HH-dd HH:mm"/>
  <property name="lenient" value="true"/>
</component>
```

The basic steps to implement the handler are as follows:

1. Create the schema file for the extension, and package it inside a bundle
2. Write a NamespaceHandler implementation and fill in the parse method
3. Export the namespace handler as a service in the service registry

5.6.1.1 Creating and packaging the schema file

The namespace handler will be packaged inside its own bundle. Inside the bundle, at location schemas/date.xsd⁴ is placed the following file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.mycompany.com/schema/myns"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:osgi="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  targetNamespace="http://www.mycompany.com/schema/date/v1.0.0"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xsd:import namespace="http://www.osgi.org/xmlns/blueprint/v1.0.0"/>
  <xsd:element name="dateformat">
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="osgi:identifiedType">
          <xsd:attribute name="lenient" type="xsd:boolean"/>
          <xsd:attribute name="pattern" type="xsd:string" use="required"/>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The bold line shows that our new dateformat element extends the basic identifiedType element from the osgi blueprint schema, meaning that it will have an id attribute.

---

⁴ The location given here is just for illustration, you can place the file anywhere inside the bundle that you choose.
5.6.1.2 NamespaceHandler implementation

The complete source code for the DateNamespaceHandler implementation is shown below:

```java
package org.osgi.service.blueprint.namespace.example;

import java.net.URL;
import java.text.SimpleDateFormat;
import org.osgi.framework.BundleContext;
import org.osgi.service.blueprint.namespace.NamespaceHandler;
import org.osgi.service.blueprint.namespace.ParserContext;
import org.osgi.service.blueprint.namespace.example.builder.MutableLocalComponentMetadata;
import org.osgi.service.blueprint.reflect.ComponentMetadata;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

/**<br>* Sample handler for a "date" namespace based on the example given here:<br>* http://static.springframework.org/spring/docs/2.5.x/reference/extensible-xml.html<br>*<br>* Handles "dateformat" elements with the following form:<br>*<br>* <date:dateformat id="dateFormat"
*   pattern="yyyy-MM-dd HH:mm"
*   lenient="true"/>
*<br>*<br>public class DateNamespaceHandler implements NamespaceHandler {

/**
 * The schema file is packaged in the same bundle as this handler class
 * for convenience, in schemas/date.xsd
 */
private static final String SCHEMA_LOCATION = "schemas/date.xsd";

private final BundleContext bundleContext;

public DateNamespaceHandler(BundleContext context) {
    this.bundleContext = context;
}

/**
 * Use the bundleContext to return a URL for accessing the schema definition file
 */
public URL getSchemaLocation(String namespace) throws IllegalArgumentException {
    return bundleContext.getBundle().getResource(SCHEMA_LOCATION);
}

/**
 * Date elements can never be used nested directly inside a component, so
 * nothing for us to do.
 */
public ComponentMetadata decorate(Node node, ComponentMetadata component,
    ParserContext context) {
    return null;
}

/**
 * Handle the "dateformat" tag (and others in time...)
*/
```
public ComponentMetadata parse(Element element, ParserContext context) {
    if (element.getLocalName().equals("dateformat")) {
        return parseDateFormat(element);
    } else {
        throw new IllegalStateException("Asked to parse unknown tag: " +
                                   element.getTagName());
    }
}

/**
 * A dataformat element defines a component, which could be a top-level named
 * component or an anonymous inner component.
 */
private ComponentMetadata parseDateFormat(Element element) {
    String name = element.hasAttribute("id") ? element.getAttribute("id") : "";
    MutableLocalComponentMetadata componentMetadata =
        new MutableLocalComponentMetadata(name, SimpleDateFormat.class.getName());

    // required attribute pattern
    String pattern = element.getAttribute("pattern");
    componentMetadata.addConstructorArg(pattern, 0);

    // this however is an optional property
    String lenient = element.getAttribute("lenient");
    if ((lenient != null) && !lenient.equals("")) {
        componentMetadata.addProperty("lenient", lenient);
    }

    return componentMetadata;
}

This implementation relies on a helper class, MutableLocalComponentMetadata that we use to create an instance of the LocalComponentMetadata interface. Future versions of this specification may standardize such implementation classes, or an equivalent builder API.

package org.osgi.service.blueprint.namespace.example.builder;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import org.osgi.service.blueprint.reflect.ComponentMetadata;
import org.osgi.service.blueprint.reflect.ConstructorInjectionMetadata;
import org.osgi.service.blueprint.reflect.LocalComponentMetadata;
import org.osgi.service.blueprint.reflect.MethodInjectionMetadata;
import org.osgi.service.blueprint.reflect.ParameterSpecification;
import org.osgi.service.blueprint.reflect.PropertyInjectionMetadata;
import org.osgi.service.blueprint.reflect.TypedStringValue;
import org.osgi.service.blueprint.reflect.Value;

/**
 * "Just enough" implementation of a mutable LocalComponentMetadata to meet the
 * needs of our namespace handler...
 */
public class MutableLocalComponentMetadata implements LocalComponentMetadata {
    private String name;
private String className;
private String initMethodName = "";
private String destroyMethodName = "";
private ComponentMetadata factoryComponent = null;
private final List<ParameterSpecification> constructorSpec =
        new ArrayList<ParameterSpecification>() { public List getParameterSpecifications() {
            return constructorSpec;
        } }; public void addConstructorArg(ParameterSpecification spec) {
        constructorSpec.add(spec);
    }

// convenience method for String-based values
public void addConstructorArg(final String value, final int index) {
    constructorSpec.add(new ParameterSpecification() {
        public int getIndex() {
            return index;
        }
        public String getTypeName() {
            return null;
        }
        public Value getValue() {
            return stringValue(value);
        }
    });
}

public Collection getPropertyInjectionMetadata() {

```java
public class MutableLocalComponentMetadata extends ComponentMetadata {  
    private String className;
    private String initMethodName = "";
    private String destroyMethodName = "";
    private ComponentMetadata factoryComponent = null;
    private final List<ParameterSpecification> constructorSpec =
            new ArrayList<ParameterSpecification>() { public List getParameterSpecifications() {
                    return constructorSpec;
                } }; public void addConstructorArg(ParameterSpecification spec) {
                    constructorSpec.add(spec);
                }

                // convenience method for String-based values
                public void addConstructorArg(final String value, final int index) {
                    constructorSpec.add(new ParameterSpecification() {
                        public int getIndex() {
                            return index;
                        }
                        public String getTypeName() {
                            return null;
                        }
                        public Value getValue() {
                            return stringValue(value);
                        }
                    });
                }

                public Collection getPropertyInjectionMetadata() {
```
return propertiesSpec;
}

public void addProperty(final String name, final Value value) {
    propertiesSpec.add(new PropertyInjectionMetadata() {
        public String getName() {
            return name;
        }
        public Value getValue() {
            return value;
        }
    });
}

public void addProperty(final String name, final String value) {
    propertiesSpec.add(new PropertyInjectionMetadata() {
        public String getName() {
            return name;
        }
        public Value getValue() {
            return stringValue(value);
        }
    });
}

public String getInitMethodName() {
    return initMethodName;
}

public void setInitMethodName(String initMethodName) {
    this.initMethodName = initMethodName;
}

public String getDestroyMethodName() {
    return destroyMethodName;
}

public void setDestroyMethodName(String destroyMethodName) {
    this.destroyMethodName = destroyMethodName;
}

public ComponentMetadata getFactoryComponent() {
    return factoryComponent;
}

public void setFactoryComponent(ComponentMetadata factoryComponent) {
    this.factoryComponent = factoryComponent;
}

public MethodInjectionMetadata getFactoryMethodMetadata() {
    return factoryMethodMetadata;
}

public void setFactoryMethodMetadata(
    MethodInjectionMetadata factoryMethodMetadata) {
    this.factoryMethodMetadata = factoryMethodMetadata;
}

public String getScope() {
    return scope;
}

public void setScope(String scope) {
    this.scope = scope;
}
public boolean isLazy() {
    return isLazy;
}

public void setLazy(boolean lazy) {
    this.isLazy = lazy;
}

public Set getExplicitDependencies() {
    return dependencies;
}

public void addDependency(String name) {
    dependencies.add(name);
}

public void removeDependency(String name) {
    dependencies.remove(name);
}

private TypedStringValue stringValue(String val) {
    return new TypedStringValue() {
        public String getStringValue() {
            return val;
        }
        public String getTypeName() {
            return null;
        }
    };
}

Such an implementation is easily generated using a modern IDE.

5.6.1.3 Registering the namespace handler

With the DateNamespaceHandler and MutableLocalComponentMetadata classes packaged up in the bundle
classpath, the last thing to do is ensure that the namespace handler is registered as a service when the bundle is
started. The easiest way to do this is to use the blueprint service. In OSGI-INF/blueprint create a file called
e.g. module-context.xml with the following declarations:

<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.osgi.org/xmlns/blueprint/v1.0.0
        http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">
    <component id="dateNamespaceHandler"
        class="org.osgi.service.blueprint.namespace.example.DateNamespaceHandler"/>

    <service ref="dateNamespaceHandler"
        interface="org.osgi.service.blueprint.namespace.NamespaceHandler">
        <service-properties>
            <entry key="org.osgi.service.blueprint.namespace">
                <array value-type="java.lang.String">
                    <value>http://www.mycompany.com/schema/date/v1.0.0</value>
                    <value>http://www.mycompany.com/schema/date</value>
                </array>
            </entry>
        </service-properties>
    </service>
</components>
5.7 Configuration Administration Service Support
The osgix namespace defines configuration elements and attributes supporting the OSGi Compendium Services. Currently the only service with dedicated support in this namespace is the Configuration Admin service.

5.7.1 Property Placeholder Support
Component property values may be sourced from the OSGi Configuration Administration service. This support is enabled via the property-placeholder element. The property placeholder element provides for replacement of delimited string values (placeholders) in component property expressions with values sourced from the configuration administration service. The required persistent-id attribute specifies the persistent identifier to be used as the key for the configuration dictionary. The default delimiter for placeholder strings is "${...}". Delimited strings can then be used for any property value of any component, and will be replaced with the configuration administration value with the given key.

Given the declarations:

```xml
<osgix:property-placeholder persistent-id="com.xyz.myapp"/>
<component id="someComponent" class="AClass">
  <property name="timeout" value="${timeout}"/>
</component>
```

Then the timeout property of someComponent will be set using the value of the timeout entry in the configuration dictionary registered under the com.xyz.myapp persistent id.

The placeholder strings are evaluated at the time that the component is instantiated. The evaluation results in a new string which is then interpreted as if it were the originally declared value. Changes to the properties made via Configuration Admin subsequent to the creation of the component do not result in re-injection of property values. See the managed-properties and managed-service-factory elements if you require this level of integration. The placeholder-prefix and placeholder-suffix attributes can be used to change the delimiter strings used for placeholder values. It is a configuration error to define multiple property-placeholder elements using the same prefix and suffix, and a ComponentDefinitionException will be thrown during context creation if such conflicting declarations are found.

It is possible to specify a default set of property values to be used in the event that the configuration dictionary does not contain an entry for a given key. The defaults-ref attribute can be used to refer to a named component of Properties or Map type. Instead of referring to an external component, the default-properties nested element may be used to define an inline set of properties.

```xml
<osgix:property-placeholder persistent-id="com.xyz.myapp">
  <osgix:default-properties>
    <property name="productCategory" value="E792"/>
    <property name="businessUnit" value="811"/>
  </osgix:default-properties>
</osgix:property-placeholder>
```

Property placeholder declarations have module context scope, and apply to any matching placeholder string regardless of the particular configuration file of the module the property placeholder and placeholder string declarations happen to be in. If a property referenced via a placeholder
definition is not defined in the configuration dictionary, and no default value has been specified, then a runtime ComponentDefinitionException will be thrown during module context creation.

The `persistent-id` attribute must refer to the persistent-id of an OSGi ManagedService, it is a configuration error to specify a factory persistent id referring to a ManagedServiceFactory.

Placeholder expressions can be used in any attribute value, as the whole or part of the value text.

### 5.7.2 Publishing Configuration Admin properties with exported services

Using the property-placeholder support it is easy to publish any named configuration-admin property as a property of a service exported to the service registry. For example:

```xml
<service interface="MyInterface" ref="MyService">
  <service-properties>
    <entry key="akey" value="${property.placeholder.key}"/>
  </service-properties>
</service>
```

To publish all of the public properties registered under a given persistent-id as properties of an exported service, without having to explicitly list all of those properties up-front, use the nested `cm-properties` element.

```xml
<service interface="org.osgi.service.cm.ManagedService" ref="MyManagedService">
  <service-properties>
    <osgix:cm-properties persistent-id="pid"/>
  </service-properties>
</service>
```

Only public properties registered under the pid (properties with a key that does not start with ".") will be published. To have the advertised service properties updated when the configuration stored under the given persistent id is updated, specify the optional `update="true"` attribute value.

### 5.7.3 Managed Properties

The `managed-properties` element can be nested inside a component declaration in order to configure component properties based on the configuration information stored under a given persistent id. It has one mandatory attribute, `persistent-id`.

An example usage of managed properties follows:

```xml
<component id="myComponent" class="AClass">
  <osgix:managed-properties persistent-id="com.xyz.messageservice"/>
  <!-- other component declarations as needed -->
</component>
```

For each key in the dictionary stored by configuration admin under the given persistent id, if the component type has a property with a matching name (following JavaBeans conventions), then that component property will be dependency injected with the value stored in configuration admin under the key.

If the definition of AClass from the example above is as follows:

```java
public class AClass {
  private int amount;
}```
public void setAmount(int amount) { this.amount = amount; }
public int getAmount() { return this.amount; }

and the configuration dictionary stored under the pid com.xyz.messageservice contains an entry "amount"=>"200", then the setAmount method will be invoked on the component instance during configuration, passing in the value 200.

If a property value is defined both in the configuration dictionary stored in the Configuration Admin service, and in a property element declaration nested in the component element, then the value from Configuration Admin takes precedence. Property values specified via property elements can therefore be treated as default values to be used if none is available through Configuration Admin.

The configuration data stored in Configuration Admin may be updated after the component has been created. By default, any updates post-creation will be ignored. To receive configuration updates, the update-strategy attribute can be used with a value of either component-managed or container-managed.

The default value of the optional update-strategy attribute is none. If an update strategy of component-managed is specified then the update-method attribute must also be used to specify the name of a method defined on the component class that will be invoked if the configuration for the component is updated. The update method must have one of the following signatures:

public void anyMethodName(Map properties)
public void anyMethodName(Map<String,?> properties); // for Java 5

When an update strategy of container-managed is specified then the container will re-inject component properties by name based on the new properties received in the update. For container-managed updates, the component class must provide setter methods for the component properties that it wishes to have updated. For each property in the updated configuration dictionary where the component class has a matching setter method, the setter method will be called with the new value.

5.7.4 Managed Service Factories

The Configuration Admin service supports a notion of a Managed Service Factory (see section 104.6 in the Compendium Specification). A managed service factory is identified by a factory pid, Configuration objects can be associated with the factory. Configuration objects associated with the factory can be added or removed at any point.

The managed-service-factory element defines a managed set of services. For each configuration object associated with the factory pid of the managed service factory, an anonymous component instance is created and registered as a service. The lifecycle of these component instances is tied to the lifecycle of the associated configuration objects. If a new configuration object is associated with the factory pid, a new component instance is created and registered as a service. If a configuration object is deleted or disassociated from the factory pid then the corresponding component instance is destroyed.

The attributes of the managed-service-factory element are:

- **id** (required)
- **factory-pid** (required) – this specifies the persistent id of the managed service factory in the Configuration Admin service
- **interface, auto-export, and ranking**, all with the same semantics as the attributes with the corresponding names defined on the <service> element. These attributes apply to each service registered on behalf of the managed service factory.
Optionally nested inside the managed-service-factory element are the interfaces, service-properties, and registration-listener elements, with the same syntax and semantics as when used nested inside of a service element.

A single nested managed-component element is required inside the managed-service-factory. The managed component declaration defines the component template for the component instances to be created and exposed as services. Managed component supports the same set of nested elements as for a component, and a subset of the attributes: class, init-method, destroy-method, factory-method and factory-component.

The signature of a destroy-method for a managed-component must follow the format:

```java
public void anyMethodName(int reasonCode);
```

where reason code is one of:

- `ModuleContext.CONFIGURATION_ADMIN_OBJECT_DELETED`
- `ModuleContext.BUNDLE_STOPPING`

To have the properties of a managed component configured from the properties stored in its associated configuration object, simply use the nested managed-properties element as in the following example. The pid for the configuration object is automatically generated by the Configuration Admin service, and is different for each managed component instance. When the managed component instance is published as a service, the `service.pid` property is set to the value of the pid for its associated configuration object. A convention is adopted that specifying an empty string for the value of the `persistent-id` attribute when used nested inside of a managed-service-factory means "the persistent id of my associated configuration object".

```xml
<managed-service-factory id="fooFactory" factory-pid="my.pid" interface="Foo">
  <managed-component class="SomeClass">
    <managed-properties persistent-id="" update-strategy="container-managed"/>
    <property name="foo" ref="someOtherComponent"/>
  </managed-component>
</managed-service-factory>
```

Given the above definition, an instance of SomeClass will be created for each configuration object associated with the managed service factory "my.pid". The instances are dependency injected with the properties found in the configuration object dictionary, and the property "foo" is also dependency injected with a reference to "someOtherComponent". Each instance is registered as a service advertising the Foo interface.

The same convention of using an empty `persistent-id` attribute value applies to the config-properties too when nested inside a managed-service-factory element. The following example will publish all of the public properties from the associated configuration object as service properties of the service published for the associated managed component.

```xml
<managed-service-factory id="fooFactory" factory-pid="my.pid" interface="Foo">
  <osgi:service-properties>
    <config-properties persistent-id=""/>
  </osgi:service-properties>
  <managed-component class="SomeClass">
    <property name="foo" ref="someOtherComponent"/>
  </managed-component>
</managed-service-factory>
```
The component defined by a managed-service-factory is of type Map<ServiceRegistration, Object> and contains one entry for each service published by it where the key is the service registration object, and the value is the service itself. The Map membership is dynamically managed as configuration objects (and hence their associated services) come and go.

A typical use case for the managed service factory element might be to publish a DataSource service for each configuration object associated with the “data.source” factory pid. Administrators can then define, configure and publish new DataSource services simply by updating configuration information in the Configuration Admin service.

### 5.7.5 Direct access to configuration data

If you need to work directly with the configuration data stored under a given persistent id or factory persistent id, the easiest way to do this is to register a service that implements either the ManagedService or ManagedServiceFactory interface and specify the pid that you are interested in as a service property. For example:

```xml
<service interface="org.osgi.service.cm.ManagedService" ref="MyManagedService">
  <service-properties>
    <entry key="service.pid" value="my.managed.service.pid"/>
  </service-properties>
</service>

<component id="myManagedService" class="com.xyz.MyManagedService"/>
```

where the class MyManagedService implements org.osgi.service.cm.ManagedService.

### 5.8 APIs

- **Local Disk**
  - `Overview`
  - `org.osgi.service.blueprint.context`
  - `org.osgi.service.blueprint.convert`
  - `org.osgi.service.blueprint.namespace`
  - `org.osgi.service.blueprint.reflect`
  - `ModuleContext`
  - `ModuleContextEventConstants`
  - `ModuleContextListener`
  - `ComponentDefinitionException`
  - `NoSuchComponentException`
  - `ServiceUnavailableException`
  - `ConversionService`
  - `Converter`
  - `ComponentDefinitionRegistry`
  - `NamespaceHandler`
  - `ParserContext`
  - `ComponentNameAlreadyInUseException`
  - `ArrayValue`
  - `BindingListenerMetadata`
  - `CollectionBasedServiceReferenceComponentMetadata`
  - `ComponentMetadata`
  - `ComponentValue`
- ConstructorInjectionMetadata
- ListValue
- LocalComponentMetadata
- MapValue
- MethodInjectionMetadata
- NullValue
- ParameterSpecification
- PropertiesValue
- PropertyInjectionMetadata
- ReferenceNameValue
- ReferenceValue
- RegistrationListenerMetadata
- ServiceExportComponentMetadata
- ServiceReferenceComponentMetadata
- SetValue
- TypedStringValue
- UnaryServiceReferenceComponentMetadata
- Value
- Constant Field Values
- Serialized Form
### Packages

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>org.osgi.service.blueprint.context</code></td>
<td>Blueprint Service Context Package Version 1.0.</td>
</tr>
<tr>
<td><code>org.osgi.service.blueprint.convert</code></td>
<td>Blueprint Service Type Conversion Package Version 1.0.</td>
</tr>
<tr>
<td><code>org.osgi.service.blueprint.namespace</code></td>
<td>Blueprint Namespace Package Version 1.0.</td>
</tr>
<tr>
<td><code>org.osgi.service.blueprint.reflect</code></td>
<td>Blueprint Reflection Package Version 1.0.</td>
</tr>
</tbody>
</table>

### 5.9 Package `org.osgi.service.blueprint.context`

Blueprint Service Context Package Version 1.0.

#### 5.9.1.1 See:

- [Description](#)
5.10 Package org.osgi.service.blueprint.context

Description

Blueprint 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. For example:

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

This package defines the primary interface to a module context, `ModuleContext`. An instance of this type is available inside a module context as an implicitly defined component with name "moduleContext".

This package also declares the supporting exception types, listener, and constants for working with a module context.

5.11 Package org.osgi.service.blueprint.convert

Blueprint Service Type Conversion Package Version 1.0.
5.11.1.1 See: Description

### Interface Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConversionService</td>
<td>Provides access to the type conversions (both predefined and user registered) that are defined for the module context</td>
</tr>
<tr>
<td>Converter</td>
<td>Implemented by type converters that extend the type conversion capabilities of a module context container.</td>
</tr>
</tbody>
</table>

5.12 Package org.osgi.service.blueprint.convert Description

Blueprint Service Type Conversion Package Version 1.0.

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

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

This package defines the Converter interface used to implement type converters, and the ConversionService interface that provides access to registered type converters. A module context contains an implicitly defined component "conversionService" that is an instance of ConversionService.

5.13 Package org.osgi.service.blueprint.namespace

Blueprint Namespace Package Version 1.0.

5.13.1.1 See: Description
### Interface Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComponentDefinitionRegistry</td>
<td>A registry of the component definitions for a given context.</td>
</tr>
<tr>
<td>NamespaceHandler</td>
<td>A namespace handler provides support for parsing custom namespace elements and attributes in module context configuration files.</td>
</tr>
<tr>
<td>ParserContext</td>
<td>A ParserContext provides contextual information to a NamespaceHandler when parsing an Element or Node from the namespace.</td>
</tr>
</tbody>
</table>

### Exception Summary

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComponentNameAlreadyInUseException</td>
<td>Exception thrown when an attempt is made to register a component with a name that is already in use by an existing component.</td>
</tr>
</tbody>
</table>

### 5.14 Package org.osgi.service.blueprint.namespace

**Description**

Blueprint Namespace Package Version 1.0.

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

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

This package provides the top-level interfaces needed for implementing a namespace handler.

### 5.15 Package org.osgi.service.blueprint.reflect

**Description**

Blueprint Reflection Package Version 1.0.
## Interface Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ArrayValue</strong></td>
<td>An array-based value.</td>
</tr>
<tr>
<td><strong>BindingListenerMetadata</strong></td>
<td>Metadata for a listener interested in service bind and unbind events for a service reference.</td>
</tr>
<tr>
<td><strong>CollectionBasedServiceReferenceComponentMetadata</strong></td>
<td>Service reference that binds to a collection of matching services from the OSGi service registry.</td>
</tr>
<tr>
<td><strong>ComponentMetadata</strong></td>
<td>Metadata for a component defined within a given module context.</td>
</tr>
<tr>
<td><strong>ComponentValue</strong></td>
<td>A value represented by an anonymous local component definition - this could be a component, reference, reference-collection or service definition.</td>
</tr>
<tr>
<td><strong>ConstructorInjectionMetadata</strong></td>
<td>Metadata describing how to instantiate a component instance by invoking one of its constructors.</td>
</tr>
<tr>
<td><strong>ListValue</strong></td>
<td>A list-based value.</td>
</tr>
<tr>
<td><strong>LocalComponentMetadata</strong></td>
<td>Metadata for a component defined locally with a module context.</td>
</tr>
<tr>
<td><strong>MapValue</strong></td>
<td>A map-based value.</td>
</tr>
<tr>
<td><strong>MethodInjectionMetadata</strong></td>
<td>Metadata describing a method to be invoked as part of component configuration.</td>
</tr>
<tr>
<td><strong>NullValue</strong></td>
<td>A value specified to be null via the element.</td>
</tr>
<tr>
<td><strong>ParameterSpecification</strong></td>
<td>Metadata describing a parameter of a method or constructor and the value that is to be passed during injection.</td>
</tr>
</tbody>
</table>
5.16 Package org.osgi.service.blueprint.reflect Description

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. For example:

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

This package provides a reflection-based view of the configuration information for a given module context. The top-level mapping between configuration elements and reflection types is as follows:

<table>
<thead>
<tr>
<th><strong>PropertiesValue</strong></th>
<th>A <code>java.util.Properties</code> based value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PropertyInjectionMetadata</strong></td>
<td>Metadata describing a property to be injected.</td>
</tr>
<tr>
<td><strong>ReferenceNameValue</strong></td>
<td>A value which represents the name of another component in the module context.</td>
</tr>
<tr>
<td><strong>ReferenceValue</strong></td>
<td>A value which refers to another component in the module context by name.</td>
</tr>
<tr>
<td><strong>RegistrationListenerMetadata</strong></td>
<td>Metadata for a listener interested in service registration and unregistration events for an exported service.</td>
</tr>
<tr>
<td><strong>ServiceExportComponentMetadata</strong></td>
<td>Metadata representing a service to be exported by a module context.</td>
</tr>
<tr>
<td><strong>ServiceReferenceComponentMetadata</strong></td>
<td>Metadata describing a reference to a service that is to be imported into the module context from the OSGi service registry.</td>
</tr>
<tr>
<td><strong>SetValue</strong></td>
<td>A set-based value.</td>
</tr>
<tr>
<td><strong>TypedStringValue</strong></td>
<td>A simple string value that will be type-converted if necessary before injecting into a target.</td>
</tr>
<tr>
<td><strong>UnaryServiceReferenceComponentMetadata</strong></td>
<td>Service reference that will bind to a single matching service in the service registry.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>A value to inject into a field, property, method argument or constructor argument.</td>
</tr>
</tbody>
</table>
LocalComponentMetadata, ComponentValue

**Configuration Element**

component reference ref-set, ref-list service constructor-arg property listener registration-listener array ref id-ref list map null props set value

**Corresponding Reflection Type(s)**


---

**org.osgi.service.blueprint.context**

### 5.17 Interface ModuleContext

```java
public interface ModuleContext {

ModuleContext providing access to the components, service exports, and service references of a module. Only bundles in the ACTIVE state may have an associated ModuleContext. A given BundleContext has at most one associated ModuleContext. An instance of ModuleContext may be obtained from within a module context by implementing the ModuleContextAware interface on a component class. Alternatively you can look up ModuleContext services in the service registry. The Constants.BUNDLE_SYMBOLICNAME and Constants.BUNDLE_VERSION service properties can be used to determine which bundle the published ModuleContext service is associated with. A ModuleContext implementation must support safe concurrent access. It is legal for the set of named components and component metadata to change between invocations on the same thread if another thread is concurrently modifying the same mutable ModuleContext implementation object.

**See Also:**

ModuleContextAware, Constants
## Field Summary

<table>
<thead>
<tr>
<th>static int</th>
<th><strong>BUNDLE_STOPPING</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>reason code for destroy method callback of a managed service factory created component, when the component is being disposed because the bundle is being stopped.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static int</th>
<th><strong>CONFIGURATION_ADMIN_OBJECT_DELETED</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>reason code for destroy method callback of a managed service factory created component, when the component is being disposed because the corresponding configuration admin object was deleted.</td>
<td></td>
</tr>
</tbody>
</table>
## Method Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>org.osgi.framework.BundleContext</code></td>
<td><code>getBundleContext()</code></td>
<td>Get the bundle context of the bundle this module context is associated with.</td>
</tr>
<tr>
<td><code>java.lang.Object</code></td>
<td><code>getComponent(String name)</code></td>
<td>Get the component instance for a given named component.</td>
</tr>
<tr>
<td><code>ComponentMetadata</code></td>
<td><code>getComponentMetadata(String name)</code></td>
<td>Get the component metadata for a given named component.</td>
</tr>
<tr>
<td><code>java.util.Set</code></td>
<td><code>getComponentNames()</code></td>
<td>The set of component names recognized by the module context.</td>
</tr>
<tr>
<td><code>java.util.Collection</code></td>
<td><code>getExportedServicesMetadata()</code></td>
<td>Get the service export metadata for every service exported by this module.</td>
</tr>
<tr>
<td><code>java.util.Collection</code></td>
<td><code>getLocalComponentsMetadata()</code></td>
<td>Get the metadata for all components defined locally within this module.</td>
</tr>
<tr>
<td><code>java.util.Collection</code></td>
<td><code>getReferencedServicesMetadata()</code></td>
<td>Get the service reference metadata for every OSGi service referenced by this module.</td>
</tr>
</tbody>
</table>
5.17.1 CONFIGURATION_ADMIN_OBJECT_DELETED

5.17.2 static final int CONFIGURATION_ADMIN_OBJECT_DELETED
reason code for destroy method callback of a managed service factory created component, when the component is being disposed because the corresponding configuration admin object was deleted.

See Also:
Constant Field Values

5.17.3 BUNDLE_STOPPING

static final int BUNDLE_STOPPING
reason code for destroy method callback of a managed service factory created component, when the component is being disposed because the bundle is being stopped.

See Also:
Constant Field Values

Method Detail

5.17.4 getComponentNames

5.17.5 java.util.Set getComponentNames()
The set of component names recognized by the module context.

Returns:
an immutable set (of Strings) containing the names of all of the components within the module.

5.17.6 getComponent

java.lang.Object getComponent(java.lang.String name)
Get the component instance for a given named component. If the component has not yet been instantiated, calling this operation will cause the component instance to be created and initialized. If the component has a prototype scope then each call to getComponent will return a new component instance. If the component has a bundle scope then the component instance returned will be the instance for the caller's bundle (and that instance will be instantiated if it has not already been created).

Note: calling getComponent from logic executing during the instantiation and configuration of a component, before the init method (if specified) has returned, may trigger a circular dependency (for a trivial example, consider a component that looks itself up by name during its init method). Implementations of the Blueprint Service are not required to support cycles in the dependency graph and may throw an exception if a cycle is detected. Implementations that can support certain kinds of cycles are free to do so.

**Parameters:**
- name - the name of the component for which the instance is to be retrieved.

**Returns:**
- the component instance, the type of the returned object is dependent on the component definition, and may be determined by introspecting the component metadata.

**Throws:**
- NoSuchComponentException - if the name specified is not the name of a component within the module.

---

### 5.17.7 getComponentMetadata

Get the component metadata for a given named component.

**Parameters:**
- name - the name of the component for which the metadata is to be retrieved.

**Returns:**
- the component metadata for the component.

**Throws:**
NoSuchComponentException - if the name specified is not the name of a component within the module.

5.17.8 getReferencedServicesMetadata

5.17.9 java.util.Collection getReferencedServicesMetadata()
Get the service reference metadata for every OSGi service referenced by this module.

Returns:
- an immutable collection of ServiceReferenceComponentMetadata, with one entry for each referenced service.

5.17.10 getExportedServicesMetadata

5.17.11 java.util.Collection getExportedServicesMetadata()
Get the service export metadata for every service exported by this module.

Returns:
- an immutable collection of ServiceExportComponentMetadata, with one entry for each service export.

5.17.12 getLocalComponentsMetadata

5.17.13 java.util.Collection getLocalComponentsMetadata()
Get the metadata for all components defined locally within this module.

Returns:
- an immutable collection of LocalComponentMetadata, with one entry for each component.

5.17.14 getBundleContext

org.osgi.framework.BundleContext getBundleContext()
Get the bundle context of the bundle this module context is associated with.

**Returns:**
the module's bundle context

---

```java
org.osgi.service.blueprint.context
Interface ModuleContextEventConstants
```

```java
public interface ModuleContextEventConstants
```

Event property names used in EventAdmin events published for a module context.
## Field Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static java. lang.String</td>
<td><strong>BUNDLE_VERSION</strong></td>
<td>The version property defining the bundle on whose behalf a module context event has been issued.</td>
</tr>
<tr>
<td>static java. lang.String</td>
<td><strong>EXTENDER_BUNDLE</strong></td>
<td>The extender bundle property defining the extender bundle processing the module context for which an event has been issued.</td>
</tr>
<tr>
<td>static java. lang.String</td>
<td><strong>EXTENDER_ID</strong></td>
<td>The extender bundle id property defining the id of the extender bundle processing the module context for which an event has been issued.</td>
</tr>
<tr>
<td>static java. lang.String</td>
<td><strong>EXTENDER_SYMBOLICNAME</strong></td>
<td>The extender bundle symbolic name property defining the symbolic name of the extender bundle processing the module context for which an event has been issued.</td>
</tr>
<tr>
<td>static java. lang.String</td>
<td><strong>TOPIC_BLUEPRINT_EVENTS</strong></td>
<td>Topic prefix for all events issued by the Blueprint Service</td>
</tr>
</tbody>
</table>
Field Detail

**BUNDLE_VERSION**

```java
static final java.lang.String BUNDLE_VERSION

The version property defining the bundle on whose behalf a module context event has been issued.

See Also:
Version, Constant Field Values
```

**EXTENDER_BUNDLE**

```java
static final java.lang.String EXTENDER_BUNDLE

The extender bundle property defining the extender bundle processing the module context for which an event has been issued.

See Also:
```
Bundle, Constant Field Values

EXTENDER_ID

static final java.lang.String EXTENDER_ID

    The extender bundle id property defining the id of the extender bundle processing
    the module context for which an event has been issued.

    See Also:
    Constant Field Values

EXTENDER_SYMBOLICNAME

static final java.lang.String EXTENDER_SYMBOLICNAME

    The extender bundle symbolic name property defining the symbolic name of the
    extender bundle processing the module context for which an event has been issued.

    See Also:
    Constant Field Values

TOPIC_BLUEPRINT_EVENTS

static final java.lang.String TOPIC_BLUEPRINT_EVENTS

    Topic prefix for all events issued by the Blueprint Service

    See Also:
    Constant Field Values

TOPIC_CREATING

static final java.lang.String TOPIC_CREATING
Topic for Blueprint Service CREATING events

See Also:
Constant Field Values

**TOPIC_CREATED**

static final java.lang.String **TOPIC_CREATED**

Topic for Blueprint Service CREATED events

See Also:
Constant Field Values

**TOPIC_DESTROYING**

static final java.lang.String **TOPIC_DESTROYING**

Topic for Blueprint Service DESTROYING events

See Also:
Constant Field Values

**TOPIC_DESTROYED**

static final java.lang.String **TOPIC_DESTROYED**

Topic for Blueprint Service DESTROYED events

See Also:
Constant Field Values

**TOPIC_WAITING**

static final java.lang.String **TOPIC_WAITING**
Topic for Blueprint Service WAITING events

See Also:
Constant Field Values

**TOPIC_FAILURE**

static final java.lang.String **TOPIC_FAILURE**

Topic for Blueprint Service FAILURE events
See Also:
Constant Field Values

```java
org.osgi.service.blueprint.context

5.18 Interface ModuleContextListener
```

public interface ModuleContextListener

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void <code>contextCreated</code></td>
<td><code>org.osgi.framework.Bundle</code> forBundle</td>
</tr>
<tr>
<td>void <code>contextCreationFailed</code></td>
<td><code>org.osgi.framework.Bundle</code> forBundle, <code>java.lang.Throwable</code> rootCause</td>
</tr>
</tbody>
</table>

### Method Detail
5.18.1 contextCreated

void contextCreated(org.osgi.framework.Bundle forBundle)

---

5.18.2 contextCreationFailed

void contextCreationFailed(org.osgi.framework.Bundle forBundle,
                         java.lang.Throwable rootCause)

---

org.osgi.service.blueprint.context

5.18.3 Class ComponentDefinitionException

java.lang.Object
  ↳ java.lang.Throwable
    ↳ java.lang.Exception
      ↳ java.lang.RuntimeException
        ↳ org.osgi.service.blueprint.context.ComponentDefinitionException

5.18.3.1 All Implemented Interfaces:
  java.io.Serializable

---

5.18.3.2 public class ComponentDefinitionException

extends java.lang.RuntimeException

Exception thrown when a configuration-related error occurs during creation of a module context.

5.18.3.3 See Also:

  Serialized Form
5.18.3.4 ComponentDefinitionException

public ComponentDefinitionException(java.lang.String explanation)

5.19 Class NoSuchComponentException

java.lang.Object
  ^java.lang.Throwable
     ^java.lang.Exception
        ^java.lang.RuntimeException
           ^org.osgi.service.blueprint.context.NoSuchComponentException

5.19.1.1 All Implemented Interfaces:
java.io.Serializable
5.19.1.2 **public class** `NoSuchComponentException`  

extends `java.lang.RuntimeException`  

Thrown when an attempt is made to lookup a component by name and no such named component exists in the module context.

5.19.1.3 **See Also:**  
[Serialized Form](#)

---

**Constructor Summary**

`NoSuchComponentException(java.lang.String componentName)`

---

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getComponentName()</code></td>
<td></td>
</tr>
<tr>
<td><code>getMessage()</code></td>
<td></td>
</tr>
</tbody>
</table>

Methods inherited from class `java.lang.Throwable`  

`fillInStackTrace`, `getCause`, `getLocalizedMessage`, `getStackTrace`, `initCause`, `printStackTrace`, `printStackTrace`, `printStackTrace`, `printStackTrace`, `setStackTrace`, `toString`

Methods inherited from class `java.lang.Object`  

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

---

**Constructor Detail**

5.19.2 **NoSuchComponentException**

`public NoSuchComponentException(java.lang.String componentName)`
## Method Detail

### 5.19.3 getComponentName

```java
public java.lang.String getComponentName()
```

### 5.19.4 getMessage

```java
public java.lang.String getMessage()
```

**5.19.4.1 Overrides:**

```
getMessage in class java.lang.Throwable
```

### org.osgi.service.blueprint.context

#### 5.19.5 Class ServiceUnavailableException

```java
java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        org.osgi.service.blueprint.context.ServiceUnavailableException
```

**5.19.5.1 All Implemented Interfaces:**

```
java.io.Serializable
```

**5.19.5.2 public class ServiceUnavailableException**

extends java.lang.RuntimeException

Thrown when an invocation is made on an OSGi service reference component, and a backing service is not available.

**5.19.5.3 See Also:**

```
Serialized Form
```
5.19.5.4 ServiceUnavailableException

public ServiceUnavailableException(java.lang.String message, java.lang.Class serviceType, java.lang.String filterExpression)
The type of the service that would have needed to be available in order for the invocation to proceed.

5.19.5.6 getFilter

public java.lang.String getFilter()

The filter expression that a service would have needed to satisfy in order for the invocation to proceed.

org.osgi.service.blueprint.convert

Interface ConversionService

public interface ConversionService

Provides access to the type conversions (both predefined and user registered) that are defined for the module context

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>convert(java.lang.Object fromValue, java.lang.Class toType)</td>
</tr>
<tr>
<td>Convert an object to an instance of the given class, using the built-in and user-registered type converters as necessary.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>convert</td>
</tr>
<tr>
<td>java.lang.Object convert(java.lang.Object fromValue, java.lang.Class toType)</td>
</tr>
<tr>
<td>throws java.lang.Exception</td>
</tr>
</tbody>
</table>
Convert an object to an instance of the given class, using the built-in and user-registered type converters as necessary.

**Parameters:**
- `fromValue` - the object to be converted
- `toType` - the type that the instance is to be converted to

**Returns:**
- an instance of the class 'toType'

**Throws:** `java.lang.Exception` - if the conversion cannot succeed. This exception is checked because callers should expect that not all source objects can be successfully converted.

---

org.osgi.service.blueprint.convert

**Interface Converter**

```java
public interface Converter
```

Implemented by type converters that extend the type conversion capabilities of a module context container.
Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>convert</code></td>
<td>java.lang.Object <code>convert(java.lang.Object source)</code></td>
</tr>
<tr>
<td></td>
<td>Convert an object to an instance of the target class.</td>
</tr>
<tr>
<td><code>getTargetClass</code></td>
<td>java.lang.Class <code>getTargetClass()</code></td>
</tr>
<tr>
<td></td>
<td>The type that this converter converts String values into.</td>
</tr>
</tbody>
</table>

Method Detail

**getTargetClass**

java.lang.Class `getTargetClass()`

The type that this converter converts String values into.

**Returns:**

Class object for the class that this converter converts to

**convert**

java.lang.Object `convert(java.lang.Object source)`

throws java.lang.Exception

Convert an object to an instance of the target class.

**Parameters:**

- **source** - the object to be converted

**Returns:**

an instance of the class returned by getTargetClass

**Throws:**

java.lang.Exception - if the conversion cannot succeed. This exception is checked because callers should expect that not all source objects can be successfully converted.
5.20 Interface ComponentDefinitionRegistry

5.20.1 public interface ComponentDefinitionRegistry

A registry of the component definitions for a given context. Implementations of ComponentDefinitionRegistry are required to support concurrent access. The state of a component registry may change between invocations on the same thread. For example, a single thread invoking containsComponentDefinition("foo") and getting a return value of 'true' may see a return value of null on a subsequent call to getComponentDefinition("foo") if another thread has removed the component definition in the meantime.

Method Summary

<table>
<thead>
<tr>
<th>boolean containsComponentDefinition(java.lang.String name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns true iff the registry contains a component definition with the given name.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ComponentMetadata getComponentDefinition(java.lang.String name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the component definition for the component with the given name.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.util.Set getComponentDefinitionNames()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the names of all the registered components.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void registerComponentDefinition(ComponentMetadata component)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register a new component definition.</td>
</tr>
</tbody>
</table>
Method Detail

5.20.2 containsComponentDefinition

```java
boolean containsComponentDefinition(java.lang.String name)
```

Returns true iff the registry contains a component definition with the given name.

5.20.3 getComponentDefinition

```java
ComponentMetadata getComponentDefinition(java.lang.String name)
```

Get the component definition for the component with the given name.

5.20.4 Returns:

the matching component definition if present, or null if no component with a matching name or alias is present.

5.20.5 getComponentDefinitionNames

```java
java.util.Set getComponentDefinitionNames()
```

Get the names of all the registered components.

5.20.6 Returns:

an immutable set (of Strings) containing the names of all registered components.
5.20.7 registerComponentDefinition

```java
void registerComponentDefinition(ComponentMetadata component)
```

Register a new component definition.

5.20.8 Throws:

- `ComponentNameAlreadyInUseException` - if the name of the component definition to be registered is already in use by an existing component definition.

5.20.9 removeComponentDefinition

```java
void removeComponentDefinition(java.lang.String name)
```

Remove a component definition from the registry. If no matching component is present then this operation does nothing.

5.20.10 Parameters:

- `name` - the name of the component to be removed.

---

org.osgi.service.blueprint.namespace

**Interface NamespaceHandler**

```java
public interface NamespaceHandler
```

A namespace handler provides support for parsing custom namespace elements and attributes in module context configuration files. It manipulates component definitions and the component registry to implement the intended semantics of the namespace. Instances of NamespaceHandler are discovered through the service registry where they should be published with a service property `org.osgi.module.context.namespace` set to the schema URI of the schema that they handle. Implementations of NamespaceHandler are required to be thread-safe.
# Method Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComponentMetadata</td>
<td><code>decoration</code> <em>(org.w3c.dom.Node node, ComponentMetadata component, ParserContext context)</em></td>
<td>Called when an attribute or nested element is encountered.</td>
</tr>
<tr>
<td>java.net.URL</td>
<td><code>getSchemaLocation</code> <em>(java.lang.String namespace)</em></td>
<td>Return the location of the schema for a given namespace.</td>
</tr>
<tr>
<td>ComponentMetadata</td>
<td><code>parse</code> <em>(org.w3c.dom.Element element, ParserContext context)</em></td>
<td>Called when a top-level (i.e.</td>
</tr>
</tbody>
</table>

## Method Detail

### `getSchemaLocation`

**java.net.URL** `getSchemaLocation` *(java.lang.String namespace)*

Return the location of the schema for a given namespace.

**Parameters:**
- namespace - one of the advertised URIs supported by this handler (as registered in the org.osgi.service.blueprint.namespace property of the service registration).

**Returns:**
- The URL where the xsd file for the schema may be found. Typically used to return a URL to a bundle resource entry so as to avoid needing to lookup schemas remotely. If null is returned then the schema location will be determined from the xsi:schemaLocation attribute value.

**Throws:**
- `java.lang.IllegalArgumentException` - if the namespace parameter is not a recognized namespace supported by this handler
parse

**ComponentMetadata** parse\(\text{org.w3c.dom.Element \ element,}\)

**ParserContext\(\text{context}\)\)

Called when a top-level (i.e. non-nested) element from the namespace is encountered. Implementers may register component definitions themselves, and/ or return a component definition to be registered.

**Parameters:**
- \text{element}- the dom element from the namespace that has just been

**decorator**

**ComponentMetadata** decorator\(\text{org.w3c.dom.Node \ node,}\)

**ComponentMetadata**\(\text{component,}\)

**ParserContext\(\text{context}\)\)

Called when an attribute or nested element is encountered. Implementors should parse the supplied Node and decorate the provided component, returning the decorated component.

**Parameters:**
- \text{node}- the dom Node from the namespace that has just been encountered
- \text{component}- the component metadata for the component in which the attribute or nested element was encountered
- \text{context}- parser context giving access component registry and context information about the current parsing location.
Returns:
the decorated component to replace the original, or simply the original component if no decoration is required.

**org.osgi.service.blueprint.namespace**

**Interface ParserContext**

```java
public interface ParserContext
```

A ParserContext provides contextual information to a NamespaceHandler when parsing an Element or Node from the namespace.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ComponentDefinitionRegistry</code></td>
<td><code>getComponentDefinitionRegistry()</code></td>
</tr>
<tr>
<td></td>
<td>The component definition registry containing all of the registered component definitions for this context</td>
</tr>
<tr>
<td><code>ComponentMetadata</code></td>
<td><code>getEnclosingComponent()</code></td>
</tr>
<tr>
<td></td>
<td>The enclosing component definition in the context of which the source node is to be processed.</td>
</tr>
<tr>
<td><code>org.w3c.dom.Node</code></td>
<td><code>getSourceNode()</code></td>
</tr>
<tr>
<td></td>
<td>The dom Node which we are currently processing</td>
</tr>
</tbody>
</table>

### Method Detail
getSourceNode

org.w3c.dom.Node getSourceNode()

The dom Node which we are currently processing

cGetComponentDefinitionRegistry

ComponentDefinitionRegistry GetComponentDefinitionRegistry()

The component definition registry containing all of the registered component definitions for this context

cGetEnclosingComponent

ComponentMetadata cGetEnclosingComponent()

The enclosing component definition in the context of which the source node is to be processed.

org.osgi.service.blueprint.namespace

5.21 Class ComponentNameAlreadyInUseException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
    org.osgi.service.blueprint.namespace.
    ComponentNameAlreadyInUseException

5.21.1 All Implemented Interfaces:

java.io.Serializable
5.21.2 public class **ComponentNameAlreadyInUseException**

extends java.lang.RuntimeException

Exception thrown when an attempt is made to register a component with a name that is already in use by an existing component.

5.21.3 See Also:

[Serialized Form](#)
### Constructor Summary

`ComponentNameAlreadyInUseException(java.lang.String name)`

### Method Summary

<table>
<thead>
<tr>
<th>java.lang.String</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getConflictingName()</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getMessage()</code></td>
</tr>
</tbody>
</table>

### Methods inherited from class java.lang.Throwable

- `fillInStackTrace`, `getCause`, `getLocalizedMessage`, `getStackTrace`, `initCause`, `printStackTrace`, `printStackTrace`, `setStackTrace`, `toString`

### Methods inherited from class java.lang.Object

- `clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `wait`, `wait`  

### Constructor Detail

#### 5.21.4 ComponentNameAlreadyInUseException

#### 5.21.5 public `ComponentNameAlreadyInUseException(java.lang.String name)`

### Method Detail
5.21.6 getMessage

    public java.lang.String getMessage() 

5.21.7 Overrides:
    getMessage in class java.lang.Throwable

5.21.8 getConflictingName

    public java.lang.String getConflictingName() 

__interface ArrayValue__

All Superinterfaces:

    Value

public interface ArrayValue

extends Value

An array-based value. Members of the array are instances of Value.
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getValueType()</td>
<td>The value-type specified for the array</td>
</tr>
<tr>
<td>getArray()</td>
<td>The array of Value objects</td>
</tr>
</tbody>
</table>

### Method Detail

**getValueType**

```java
java.lang.String getValueType()
```

The value-type specified for the array

**getArray**

```java
Value[] getArray()
```

The array of Value objects

---

**org.osgi.service.blueprint.reflect**

**Interface BindingListenerMetadata**

```java
public interface BindingListenerMetadata
```

Metadata for a listener interested in service bind and unbind events for a service reference.
The component instance that will receive bind and unbind events. The returned value must reference a component and therefore be either a ComponentValue, ReferenceValue, or ReferenceNameValue.

**Returns:**
the listener component reference.

**getBindMethodName**

```java
java.lang.String getBindMethodName()
```

The name of the method to invoke on the listener component when a matching service is bound to the reference.

**Returns:**
the bind callback method name.
getUnbindMethodName

java.lang.String getUnbindMethodName()

The name of the method to invoke on the listener component when a service is unbound from the reference.

Returns:
the unbind callback method name.

5.21.8.1 org.osgi.service.blueprint.reflect

Interface
CollectionBasedServiceReferenceComponentMetadata

5.21.9 All Superinterfaces:
ComponentMetadata, ServiceReferenceComponentMetadata

5.21.10 public interface

CollectionBasedServiceReferenceComponentMetadata

extends ServiceReferenceComponentMetadata

Service reference that binds to a collection of matching services from the OSGi service registry.
### Field Summary

<table>
<thead>
<tr>
<th>static int</th>
<th>MEMBER_TYPE_SERVICE_REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Collection contains service references</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static int</th>
<th>MEMBER_TYPE_SERVICES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Collection contains service instances</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static int</th>
<th>ORDER_BASIS_SERVICE_REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Create ordering based on comparison of service reference objects.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static int</th>
<th>ORDER_BASIS_SERVICES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Create ordering based on comparison of service objects.</td>
</tr>
</tbody>
</table>

**Fields inherited from interface org.osgi.service.blueprint.reflect.ServiceReferenceComponentMetadata**

MANDATORY_AVAILABILITY, OPTIONAL_AVAILABILITY
Method Summary

<table>
<thead>
<tr>
<th>java.lang.Class</th>
<th>getCollectionType()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>getComparator()</td>
</tr>
<tr>
<td>int</td>
<td>getMemberType()</td>
</tr>
<tr>
<td>int</td>
<td>getOrderingComparisonBasis()</td>
</tr>
</tbody>
</table>

Create ordering based on comparison of service objects.

Methods inherited from interface org.osgi.service.blueprint.reflect.ServiceReferenceComponentMetadata

getBindingListeners, getComponentName, getFilter, getInterfaceNames, getServiceAvailabilitySpecification

Methods inherited from interface org.osgi.service.blueprint.reflect.ComponentMetadata

getExplicitDependencies, getName

Field Detail

5.21.11 ORDER_BASIS_SERVICES

static final int ORDER_BASIS_SERVICES

Create ordering based on comparison of service objects.

5.21.12 See Also:

Constant Field Values
5.21.13 ORDER_BASIS_SERVICE_REFERENCES
static final int ORDER_BASIS_SERVICE_REFERENCES

Create ordering based on comparison of service reference objects.

5.21.14 See Also:
Constant Field Values

5.21.15 MEMBER_TYPE_SERVICES
static final int MEMBER_TYPE_SERVICES

Collection contains service instances

5.21.16 See Also:
Constant Field Values

5.21.17 MEMBER_TYPE_SERVICE_REFERENCES
static final int MEMBER_TYPE_SERVICE_REFERENCES

Collection contains service references

5.21.18 See Also:
Constant Field Values

Method Detail

5.21.19 getCollectionType
java.lang.Class getCollectionType()

The type of collection to be created.

5.21.20 Returns:
Class object for the specified collection type (List, Set).
5.21.21 getComparator

```java
Value getComparator()
```

The comparator specified for ordering the collection, or null if no comparator was specified.

**5.21.22 Returns:**
if a comparator was specified then a Value object identifying the comparator (a ComponentValue, ReferenceValue, or ReferenceNameValue) is returned. If no comparator was specified then null will be returned.

5.21.23 getOrderingComparisonBasis

```java
int getOrderingComparisonBasis()
```

The basis on which to perform ordering, if specified.

**5.21.24 Returns:**
one of ORDER_BASIS_SERVICES and ORDER_BASIS_SERVICE_REFERENCES

5.21.25 getMemberType

```java
int getMemberType()
```

Whether the collection will contain service instances, or service references

---

**org.osgi.service.blueprint.reflect**

**Interface ComponentMetadata**

**All Known Subinterfaces:**
- CollectionBasedServiceReferenceComponentMetadata
- LocalComponentMetadata
- ServiceExportComponentMetadata
- ServiceReferenceComponentMetadata
- UnaryServiceReferenceComponentMetadata
public interface ComponentMetadata

Metadata for a component defined within a given module context.

See Also:

LocalComponentMetadata, ServiceReferenceComponentMetadata, ServiceExportComponentMetadata

### Method Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.Set</td>
<td>getExplicitDependencies()</td>
<td>The names of any components listed in a &quot;depends-on&quot; attribute for this component.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getName()</td>
<td>The name of the component.</td>
</tr>
</tbody>
</table>

### Method Detail

**getName**

```java
java.lang.String getName()
```

The name of the component.

**Returns:**
component name. The component name may be null if this is an anonymously defined inner component.

**getExplicitDependencies**

```java
java.util.Set getExplicitDependencies()
```
The names of any components listed in a "depends-on" attribute for this component.

**Returns:**
an immutable set of component names for components that we have explicitly declared a dependency on, or an empty set if none.

---

```java
org.osgi.service.blueprint.reflect

Interface ComponentValue
```

**All Superinterfaces:**

- Value

```java
public interface ComponentValue
extends Value
```

A value represented by an anonymous local component definition - this could be a component, reference, reference-collection or service definition.

---

**Method Summary**

<table>
<thead>
<tr>
<th>ComponentMetadata</th>
<th>getComponentMetadata()</th>
</tr>
</thead>
</table>

**Method Detail**

`getComponentMetadata`

```java
ComponentMetadata getComponentMetadata()
```
org.osgi.service.blueprint.reflect

Interface ConstructorInjectionMetadata

public interface ConstructorInjectionMetadata

Metadata describing how to instantiate a component instance by invoking one of its constructors.

Method Summary

<table>
<thead>
<tr>
<th>java.util.List</th>
<th>getParameterSpecifications()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The parameter specifications that determine which constructor to invoke and what arguments to pass to it.</td>
</tr>
</tbody>
</table>

Method Detail

g.getParameterSpecifications

java.util.List getParameterSpecifications()

The parameter specifications that determine which constructor to invoke and what arguments to pass to it.

Returns:
an immutable list of ParameterSpecification, or an empty list if the default constructor is to be invoked. The list is ordered by ascending parameter index. I.e., the first parameter is first in the list, and so on.

org.osgi.service.blueprint.reflect

Interface ListValue
All Superinterfaces:
   Value

public interface ListValue

extends Value

A list-based value. Members of the List are instances of Value.

Method Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.List</td>
<td>getList()</td>
<td>The List (of Value objects) for this List-based value</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getValueType()</td>
<td>The value-type specified for the list elements, or null if none given</td>
</tr>
</tbody>
</table>

Method Detail

getValueType

java.lang.String getValueType()

The value-type specified for the list elements, or null if none given

getList

java.util.List getList()

The List (of Value objects) for this List-based value
org.osgi.service.blueprint.reflect

5.22 Interface LocalComponentMetadata
All Superinterfaces:
   ComponentMetadata

5.22.1 public interface LocalComponentMetadata
   extends ComponentMetadata

Metadata for a component defined locally with a module context.

| Field Summary | | |
|---------------|-----------------------|
| static java.lang.String | SCOPE_BUNDLE |
| static java.lang.String | SCOPE_PROTOTYPE |
| static java.lang.String | SCOPE_SINGLETON |

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>getClass()</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>The name of the class type specified for this component.</td>
</tr>
<tr>
<td>ConstructorInjectionMetadata</td>
<td>getConstructorInjectionMetadata()</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td></td>
<td>The constructor injection metadata for this component.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getDestroyMethodName()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The name of the destroy method specified for this component, if any.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>getFactoryComponent()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The component instance on which to invoke the factory method (if specified).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MethodInjectionMetadata</th>
<th>getFactoryMethodMetadata()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The metadata describing how to create the component instance by invoking a method (as opposed to a constructor) if factory methods are used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getInitMethodName()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The name of the init method specified for this component, if any.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.util.Collection</th>
<th>getPropertyInjectionMetadata()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The property injection metadata for this component.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getScope()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The specified scope for the component lifecycle.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>isLazy()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Is this component to be lazily instantiated?</td>
</tr>
</tbody>
</table>

Methods inherited from interface org.osgi.service.blueprint.reflect.ComponentMetadata
getExplicitDependencies, getName
### Field Detail

#### 5.22.2 SCOPE_SINGLETON

static final java.lang.String SCOPE_SINGLETON

See Also:
- [Constant Field Values](#)

---

#### 5.22.3 SCOPE_PROTOTYPE

static final java.lang.String SCOPE_PROTOTYPE

See Also:
- [Constant Field Values](#)

---

#### 5.22.4 SCOPE_BUNDLE

static final java.lang.String SCOPE_BUNDLE

See Also:
- [Constant Field Values](#)

### Method Detail

#### 5.22.5 getClassName

java.lang.String getClassName()

The name of the class type specified for this component.

**Returns:**
the name of the component class. If no class was specified in the component definition (because the a factory component is used instead) then this method will return null.

---

#### 5.22.6 getInitMethodName

java.lang.String getInitMethodName()
The name of the init method specified for this component, if any.

**Returns:**
the method name of the specified init method, or null if no init method was specified.

---

### 5.22.7 `getDestroyMethodName`

```java
java.lang.String getDestroyMethodName()
```

The name of the destroy method specified for this component, if any.

**Returns:**
the method name of the specified destroy method, or null if no destroy method was specified.

---

### 5.22.8 `getConstructorInjectionMetadata`

```java
ConstructorInjectionMetadata getConstructorInjectionMetadata()
```

The constructor injection metadata for this component.

**Returns:**
the constructor injection metadata. This is guaranteed to be non-null and will refer to the default constructor if no explicit constructor injection was specified for the component.

---

### 5.22.9 `getPropertyInjectionMetadata`

```java
java.util.Collection getPropertyInjectionMetadata()
```

The property injection metadata for this component.

**Returns:**
an immutable collection of PropertyInjectionMetadata, with one entry for each property to be injected. If no property injection was specified for this component then an empty collection will be returned.
5.22.10 **isLazy**

```java
boolean isLazy()
```

Is this component to be lazily instantiated?

**Returns:**

true, iff this component definition specifies lazy instantiation.

5.22.11 **getFactoryMethodMetadata**

```java
MethodInjectionMetadata getFactoryMethodMetadata()
```

The metadata describing how to create the component instance by invoking a method (as opposed to a constructor) if factory methods are used.

**Returns:**

the method injection metadata for the specified factory method, or null if no factory method is used for this component.

5.22.12 **getFactoryComponent**

```java
Value getFactoryComponent()
```

The component instance on which to invoke the factory method (if specified).

**Returns:**

when a factory method and factory component has been specified for this component, this operation returns the metadata specifying the component on which the factory method is to be invoked. When no factory component has been specified this operation will return null. A return value of null with a non-null factory method indicates that the factory method should be invoked as a static method on the component class itself. For a non-null return value, the Value object returned will be either a ComponentValue or ReferenceValue.

5.22.13 **getScope**

```java
java.lang.String getScope()
```
The specified scope for the component lifecycle.

Returns:
   a String indicating the scope specified for the component.

See Also:
   SCOPE_SINGLETON, SCOPE_PROTOTYPE, SCOPE_BUNDLE

---

org.osgi.service.blueprint.reflect

Interface MapValue

All Superinterfaces:
   Value

public interface MapValue

   extends Value

A map-based value. Map keys are instances of Value, as are the Map entry values themselves.
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getKeyType()</code></td>
<td>java.lang.String get<strong>KeyType</strong>() The key-type specified for map keys, or null if none given</td>
</tr>
<tr>
<td><code>getKeyType()</code></td>
<td>java.lang.String get<strong>KeyType</strong>() The key-type specified for map keys, or null if none given</td>
</tr>
<tr>
<td><code>getMap()</code></td>
<td>java.util.Map get<strong>Map</strong>() The Map of Value-&gt;Value mappings for this map-based value</td>
</tr>
<tr>
<td><code>getValueType()</code></td>
<td>java.lang.String get<strong>Value</strong>Type() The value-type specified for map values, or null if none given</td>
</tr>
<tr>
<td><code>getValueType()</code></td>
<td>java.lang.String get<strong>Value</strong>Type() The value-type specified for map values, or null if none given</td>
</tr>
</tbody>
</table>

### Method Detail

**getValueType**

```java
java.lang.String getValueType()  
```

The value-type specified for map values, or null if none given

**getKeyType**

```java
java.lang.String getKeyType()  
```

The key-type specified for map keys, or null if none given

**getMap**

```java
java.util.Map getMap()  
```

The Map of Value->Value mappings for this map-based value
```
org.osgi.service.blueprint.reflect

Interface MethodInjectionMetadata

public interface MethodInjectionMetadata

Metadata describing a method to be invoked as part of component configuration.

Method Summary

java.lang.String getName()

The name of the method to be invoked.

java.util.List getParameterSpecifications()

The parameter specifications that determine which method to invoke (in the case of overloading) and what arguments to pass to it.

Method Detail

getName

java.lang.String getName()

The name of the method to be invoked.

Returns:

the method name, overloaded methods are disambiguated by parameter
```
getParameterSpecifications

java.util.List getParameterSpecifications()

The parameter specifications that determine which method to invoke (in the case of overloading) and what arguments to pass to it.

**Returns:**

an immutable List of ParameterSpecification, or an empty list if the method takes no arguments. The list is ordered by ascending parameter index. I.e., the first parameter is first in the list, and so on.

---

## Interface NullableValue

**All Superinterfaces:**

Value

---

public interface NullableValue

extends Value

A value specified to be null via the element.
Field Summary

| static final NULLValue | NULL |

Field Detail

NULL

null final NULLValue NULL

org.osgi.service.blueprint.reflect

Interface ParameterSpecification

public interface ParameterSpecification

Metadata describing a parameter of a method or constructor and the value that is to be passed during injection.
### Method Summary

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><code>getIndex()</code></td>
<td>The (zero-based) index into the parameter list of the method or constructor to be invoked for this parameter.</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td><code>getTypeName()</code></td>
<td>The type to convert the value into when invoking the constructor or factory method. If no explicit type was specified on the component definition then this method returns null.</td>
</tr>
<tr>
<td><code>Value</code></td>
<td><code>getValue()</code></td>
<td>The value to inject into the parameter.</td>
</tr>
</tbody>
</table>

### Method Detail

**getValue**

```java
public Value getValue()
```

The value to inject into the parameter.

**Returns:**

the parameter value

**getTypeName**

```java
public java.lang.String getTypeName()
```

The type to convert the value into when invoking the constructor or factory method. If no explicit type was specified on the component definition then this method returns null.

**Returns:**

the explicitly specified type to convert the value into, or null if no type was specified in the component definition.
getIndex

int getIntIndex()

The (zero-based) index into the parameter list of the method or constructor to be
invoked for this parameter. This is determined either by explicitly specifying the
index attribute in the component declaration, or by declaration order of constructor-
arg elements if the index was not explicitly set.

**Returns:**
the zero-based parameter index

---

org.osgi.service.blueprint.reflect

**Interface PropertiesValue**

**All Superinterfaces:**
Value

```java
public interface PropertiesValue
extends Value
A java.util.Properties based value
```

---

**Method Summary**

```java
java.util.Properties
getPropertiesValue()
```

---

**Method Detail**
getPropertiesValue

java.util.Properties getPropertiesValue()
**getValue**

```
Value getValue()
```

The value to inject the property with.

**Returns:**

the property value.

---

**org.osgi.service.blueprint.reflect**

**Interface ReferenceNameValue**

**All Superinterfaces:**

`Value`

**public interface ReferenceNameValue**

extends `Value`

A value which represents the name of another component in the module context. The name itself will be injected, not the component that the name refers to.

---

**Method Summary**

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th><code>getReferenceName()</code></th>
</tr>
</thead>
</table>

**Method Detail**
getReferenceName

java.lang.String getReferenceName()

---

org.osgi.service.blueprint.reflect

Interface ReferenceValue

All Superinterfaces:
  Value

public interface ReferenceValue
extends Value

A value which refers to another component in the module context by name.

---

Method Summary

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getComponentName()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The name of the referenced component.</td>
</tr>
</tbody>
</table>

---

Method Detail

gGetComponentName

java.lang.String getComponentName()

The name of the referenced component.
**Interface RegistrationListenerMetadata**

```java
public interface RegistrationListenerMetadata
```

Metadata for a listener interested in service registration and unregistration events for an exported service.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getListenerComponent()</code></td>
<td>The component instance that will receive registration and unregistration events.</td>
</tr>
<tr>
<td><code>getRegistrationMethodName()</code></td>
<td>The name of the method to invoke on the listener component when the exported service is registered with the service registry.</td>
</tr>
<tr>
<td><code>getUnregistrationMethodName()</code></td>
<td>The name of the method to invoke on the listener component when the exported service is unregistered from the service registry.</td>
</tr>
</tbody>
</table>

### Method Detail

**getListenerComponent**

```java
Value getListenerComponent()
```

The component instance that will receive registration and unregistration events. The returned value must reference a component and therefore be either a `ComponentValue`, `ReferenceValue`, or `ReferenceNameValue`.

**Returns:**
the listener component reference.

---

**getRegistrationMethodName**

```java
java.lang.String getRegistrationMethodName()
```

The name of the method to invoke on the listener component when the exported service is registered with the service registry.

**Returns:**

the registration callback method name.

---

**getUnregistrationMethodName**

```java
java.lang.String getUnregistrationMethodName()
```

The name of the method to invoke on the listener component when the exported service is unregistered from the service registry.

**Returns:**

the unregistration callback method name.

---

**org.osgi.service.blueprint.reflect**

**Interface ServiceExportComponentMetadata**

**All Superinterfaces:**

- `ComponentMetadata`

---

```java
public interface ServiceExportComponentMetadata
```
extends ComponentMetadata

Metadata representing a service to be exported by a module context.
### Field Summary

<table>
<thead>
<tr>
<th>static int</th>
<th><strong>EXPORT_MODE_ALL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Advertise all Java classes and interfaces in the exported component's type as service interfaces.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static int</th>
<th><strong>EXPORT_MODE_CLASS_HIERARCHY</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Advertise all Java classes in the hierarchy of the exported component's type as service interfaces.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static int</th>
<th><strong>EXPORT_MODE_DISABLED</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Do not auto-detect types for advertised service interfaces</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static int</th>
<th><strong>EXPORT_MODE_INTERFACES</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Advertise all Java interfaces implemented by the exported component as service interfaces.</td>
</tr>
</tbody>
</table>
### Method Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><code>getAutoExportMode()</code></td>
<td>Return the auto-export mode specified.</td>
</tr>
<tr>
<td>Value</td>
<td><code>getExportedComponent()</code></td>
<td>The component that is to be exported as a service.</td>
</tr>
<tr>
<td><code>java.util.Set</code></td>
<td><code>getInterfaceNames()</code></td>
<td>The type names of the set of interface types that the service should be advertised as supporting, as specified in the component declaration.</td>
</tr>
<tr>
<td>int</td>
<td><code>getRanking()</code></td>
<td>The ranking value to use when advertising the service.</td>
</tr>
<tr>
<td><code>java.util.Collection</code></td>
<td><code>getRegistrationListeners()</code></td>
<td>The listeners that have registered to be notified when the exported service is registered and unregistered with the framework.</td>
</tr>
<tr>
<td><code>java.util.Map</code></td>
<td><code>getServiceProperties()</code></td>
<td>The user declared properties to be advertised with the service.</td>
</tr>
</tbody>
</table>

**Methods inherited from interface org.osgi.service.blueprint.reflect.ComponentMetadata**

- `getExplicitDependencies`, `getName`

### Field Detail

**EXPORT_MODE_DISABLED**

- `static final int EXPORT_MODE_DISABLED`

  Do not auto-detect types for advertised service interfaces

**See Also:**

- [Constant Field Values](#)
EXPORT_MODE_INTERFACES

```java
static final int EXPORT_MODE_INTERFACES

Advertise all Java interfaces implemented by the exported component as service interfaces.

See Also:
- Constant Field Values
```

EXPORT_MODE_CLASS_HIERARCHY

```java
static final int EXPORT_MODE_CLASS_HIERARCHY

Advertise all Java classes in the hierarchy of the exported component's type as service interfaces.

See Also:
- Constant Field Values
```

EXPORT_MODE_ALL

```java
static final int EXPORT_MODE_ALL

Advertise all Java classes and interfaces in the exported component's type as service interfaces.

See Also:
- Constant Field Values
```
getExportedComponent

Value getExportedComponent()

The component that is to be exported as a service. Value must refer to a component and therefore be either a ComponentValue or ReferenceValue.

Returns:
the component to be exported as a service.

getInterfaceNames

java.util.Set getInterfaceNames()

The type names of the set of interface types that the service should be advertised as supporting, as specified in the component declaration.

Returns:
an immutable set of (String) type names, or an empty set if using auto-export

getAutoExportMode

int getAutoExportMode()

Return the auto-export mode specified.

Returns:
One of EXPORT_MODE_DISABLED,
EXPORT_MODE_INTERFACES,
EXPORT_MODE_CLASS_HIERARCHY, EXPORT_MODE_ALL

getServiceProperties

java.util.Map getServiceProperties()

The user declared properties to be advertised with the service.
**Returns:**
Map containing the set of user declared service properties (may be empty if no properties were specified).

### getRanking

```java
int getRanking()
```

The ranking value to use when advertising the service.

**Returns:**
service ranking

### getRegistrationListeners

```java
java.util.Collection getRegistrationListeners()
```

The listeners that have registered to be notified when the exported service is registered and unregistered with the framework.

**Returns:**
an immutable collection of RegistrationListenerMetadata

---

**org.osgi.service.blueprint.reflect**

**Interface ServiceReferenceComponentMetadata**

**All Superinterfaces:**
ComponentMetadata

**All Known Subinterfaces:**
CollectionBasedServiceReferenceComponentMetadata
UnaryServiceReferenceComponentMetadata
public interface ServiceReferenceComponentMetadata extends ComponentMetadata

Metadata describing a reference to a service that is to be imported into the module context from the OSGi service registry.
### Field Summary

<table>
<thead>
<tr>
<th>static int</th>
<th><strong>MANDATORY_Availability</strong></th>
<th>A matching service is required at all times.</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int</td>
<td><strong>OPTIONAL_Availability</strong></td>
<td>A matching service is not required to be present.</td>
</tr>
</tbody>
</table>

### Method Summary
<table>
<thead>
<tr>
<th>java.util.Collection</th>
<th>getBindingListeners()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The set of listeners registered to receive bind and unbind events for backing services.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getComponentName()</td>
</tr>
<tr>
<td></td>
<td>The value of the component name attribute, if specified.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getFilter()</td>
</tr>
<tr>
<td></td>
<td>The filter expression that a matching service must pass</td>
</tr>
<tr>
<td>java.util.Set</td>
<td>getInterfaceNames()</td>
</tr>
<tr>
<td></td>
<td>The interface types that the matching service must support</td>
</tr>
<tr>
<td>int</td>
<td>getServiceAvailabilitySpecification()</td>
</tr>
<tr>
<td></td>
<td>Whether or not a matching service is required at all times.</td>
</tr>
</tbody>
</table>

Methods inherited from interface org.osgi.service.blueprint.reflect.ComponentMetadata

getExplicitDependencies, getName

Field Detail

**MANDATORY_AVAILABILITY**

**static final int MANDATORY_AVAILABILITY**

A matching service is required at all times.

See Also:  
Constant Field Values

**OPTIONAL_AVAILABILITY**
static final int OPTIONAL_AVAILABILITY

A matching service is not required to be present.

See Also: Constant Field Values

Method Detail

getServiceAvailabilitySpecification

int getServiceAvailabilitySpecification()  

Whether or not a matching service is required at all times.

Returns:

one of MANDATORY_AVAILABILITY or OPTIONAL_AVAILABILITY

getInterfaceNames

java.util.Set getInterfaceNames()  

The interface types that the matching service must support

Returns:

an immutable set of type names

getComponentName

java.lang.String getComponentName()  

The value of the component name attribute, if specified.

Returns:

the component name attribute value, or null if the attribute was not specified
getFilter

java.lang.String getFilter()

The filter expression that a matching service must pass

Returns:
filter expression

getBindingListeners

java.util.Collection getBindingListeners()

The set of listeners registered to receive bind and unbind events for backing services.

Returns:
an immutable collection of registered BindingListenerMetadata

Interface SetValue

All Superinterfaces:
Value

public interface SetValue

extends Value

A set-based value. Members of the set are instances of Value.
### Method Summary

<table>
<thead>
<tr>
<th>java.util.Set</th>
<th><strong>getSet()</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The Set (of Value objects) for this set-based value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th><strong>getValueType()</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The value-type specified for the set elements, or null if none given</td>
</tr>
</tbody>
</table>

### Method Detail

**getValueType**

```java
java.lang.String getValueType()
```

The value-type specified for the set elements, or null if none given

**getSet**

```java
java.util.Set getSet()
```

The Set (of Value objects) for this set-based value

---

**org.osgi.service.blueprint.reflect**

**Interface TypedStringValue**

All Superinterfaces:

```
Value
```

```
public interface TypedStringValue
```
extends Value

A simple string value that will be type-converted if necessary before injecting into a target.

### Method Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td><code>getStringValue()</code></td>
<td>The string value (unconverted) of this value.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td><code>getTypeName()</code></td>
<td>The name of the type to which this value should be coerced.</td>
</tr>
</tbody>
</table>

### Method Detail

#### getStringValue

```java
java.lang.String `getStringValue()`
```

The string value (unconverted) of this value.

#### getTypeName

```java
java.lang.String `getTypeName()`
```

The name of the type to which this value should be coerced. May be null.

---

org.osgi.service.blueprint.reflect

Interface `UnaryServiceReferenceComponentMetadata`
All Superinterfaces:
   ComponentMetadata, ServiceReferenceComponentMetadata

public interface UnaryServiceReferenceComponentMetadata
extends ServiceReferenceComponentMetadata

Service reference that will bind to a single matching service in the service registry.
Field Summary

Fields inherited from interface org.osgi.service.blueprint.reflect. ServiceReferenceComponentMetadata

MANDATORY_AVAILABILITY, OPTIONAL_AVAILABILITY

Method Summary

long `getTimeout()`

Timeout for service invocations when a matching backing service is unavailable.

Methods inherited from interface org.osgi.service.blueprint.reflect. ServiceReferenceComponentMetadata

`getBindingListeners`, `getComponentName`, `getFilter`, `getInterfaceNames`, `getServiceAvailabilitySpecification`

Methods inherited from interface org.osgi.service.blueprint.reflect. ComponentMetadata

`getExplicitDependencies`, `getName`

Method Detail

`getTimeout`

`long getTimeout()`

Timeout for service invocations when a matching backing service is unavailable.
Returns:
   service invocation timeout in milliseconds

---

`org.osgi.service.blueprint.reflect`  
**Interface Value**

**All Known Subinterfaces:**  
`ArrayValue`, `ComponentValue`, `ListValue`, `MapValue`, `NullValue`, `PropertiesValue`, `ReferenceNameValue`, `ReferenceValue`, `SetValue`, `TypedStringValue`

**public interface Value**

A value to inject into a field, property, method argument or constructor argument.

---

**Constant Field Values**

5.22.14 Contents

- `org.osgi.*`
### org.osgi.*

#### org.osgi.service.blueprint.context.ModuleContext

<table>
<thead>
<tr>
<th>Public Static Final Int</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BundleStopping</td>
<td>BUNDLE_STOPPING</td>
<td></td>
</tr>
<tr>
<td>ConfigurationAdminObjectDeleted</td>
<td>CONFIGURATION_ADMIN_OBJECT_DELETED</td>
<td></td>
</tr>
</tbody>
</table>

#### org.osgi.service.blueprint.context.ModuleContextEventConstants

<table>
<thead>
<tr>
<th>Public Static Final Java Lang String</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BundleVersion</td>
<td>BUNDLE_VERSION</td>
<td>&quot;bundle.version&quot;</td>
</tr>
<tr>
<td>ExtenderBundle</td>
<td>EXTENDER_BUNDLE</td>
<td>&quot;extender.bundle&quot;</td>
</tr>
<tr>
<td>ExtenderId</td>
<td>EXTENDER_ID</td>
<td>&quot;extender.bundle.id&quot;</td>
</tr>
<tr>
<td>ExtenderSymbolicName</td>
<td>EXTENDER_SYMBOLICNAME</td>
<td>&quot;extender.bundle.symbolicName&quot;</td>
</tr>
<tr>
<td>TopicBlueprintEvents</td>
<td>TOPIC_BLUEPRINT_EVENTS</td>
<td>&quot;org.osgi/service/blueprint&quot;</td>
</tr>
</tbody>
</table>
| public static final java.lang.String | TOPIC_CREATED | "org/osgi/service/blueprint/context/CREATED"
| public static final java.lang.String | TOPIC_CREATING | "org/osgi/service/blueprint/context/CREATING"
| public static final java.lang.String | TOPIC_DESTROYED | "org/osgi/service/blueprint/context/DESTROYED"
| public static final java.lang.String | TOPIC_DESTRUCTING | "org/osgi/service/blueprint/context/DESTROYING"
| public static final java.lang.String | TOPIC_FAILURE | "org/osgi/service/blueprint/context/FAILURE"
| public static final java.lang.String | TOPIC_WAITING | "org/osgi/service/blueprint/context/WAITING"

org.osgi.service.blueprint.reflect.
CollectionBasedServiceReferenceComponentMetadata

| public static final int | MEMBER_TYPE_SERVICE_REFERENCES | 2
<table>
<thead>
<tr>
<th>public static final int</th>
<th>org.osgi.service.blueprint.reflect.LocalComponentMetadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMBER_TYPE_SERVICES</td>
<td>Scope: Bundle, &quot;bundle&quot;</td>
</tr>
<tr>
<td>ORDER_BASIS_SERVICE_REFERENCES</td>
<td>Scope: Prototype, &quot;prototype&quot;</td>
</tr>
<tr>
<td>ORDER_BASIS_SERVICES</td>
<td>Scope: Singleton, &quot;singleton&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final int</th>
<th>org.osgi.service.blueprint.reflect.ServiceExportComponentMetadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPORT_MODE_ALL</td>
<td>4</td>
</tr>
<tr>
<td>EXPORT_MODE_CLASS_HIERARCHY</td>
<td>3</td>
</tr>
<tr>
<td>EXPORT_MODE_DISABLED</td>
<td>1</td>
</tr>
<tr>
<td>EXPORT_MODE_INTERFACES</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final int</th>
<th>org.osgi.service.blueprint.reflect.ServiceReferenceComponentMetadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>MANDATORY_AVAILABILITY</td>
<td>1</td>
</tr>
<tr>
<td>OPTIONAL_AVAILABILITY</td>
<td>2</td>
</tr>
</tbody>
</table>

**Serialized Form**
Package org.osgi.service.blueprint.context

Class org.osgi.service.blueprint.context.
ComponentDefinitionException extends java.lang.
RuntimeException implements Serializable

Class org.osgi.service.blueprint.context.
NoSuchComponentException extends java.lang.
RuntimeException implements Serializable

Serialized Fields

cOMPONENTNAME

java.lang.String componentName

Class org.osgi.service.blueprint.context.
ServiceUnavailableException extends java.lang.
RuntimeException implements Serializable

Serialized Fields

_SERVICETYPE

java.lang.Class serviceType

filter
java.lang.String filter

Package org.osgi.service.blueprint.namespace

Class org.osgi.service.blueprint.namespace.ComponentNameAlreadyInUseException extends java.lang.RuntimeException implements Serializable

Serialized Fields
duplicateName

java.lang.String duplicateName

5.23 '.osgi' Schema

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<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="1.0.0">

<!-- Schema elements for core component declarations -->

<xsd:complexType name="TidentifiedType" abstract="true">
    <xsd:attribute name="id" type="xsd:ID">
    </xsd:attribute>
</xsd:complexType>

<xsd:element name="components">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="description" minOccurs="0"/>
            <xsd:element ref="type-converters" minOccurs="0" maxOccurs="1"/>
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                ...
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```
<xsd:element ref="component"/>
<xsd:element ref="ref-list"/>
<xsd:element ref="ref-set"/>
<xsd:element ref="reference"/>
<xsd:element ref="service"/>
<xsd:any namespace="##other" processContents="strict" minOccurs="0" maxOccurs="unbounded"/>
</xsd:choice>
</xsd:sequence>
<xsd:attribute name="default-lazy-init" default="false" type="xsd:boolean"/>
<xsd:attribute name="default-init-method" type="xsd:string"/>
<xsd:attribute name="default-destroy-method" type="xsd:string"/>
<xsd:attributeGroup ref="defaults"/>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:element>

<xsd:element name="description">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="component"/>
      <xsd:element ref="ref"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

<xsd:element name="type-converters">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="component"/>
      <xsd:element ref="ref"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

<xsd:group name="componentElements">
  <xsd:sequence>
    <xsd:element ref="description" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="constructor-arg"/>
      <xsd:element ref="property"/>
    </xsd:choice>
    <xsd:any namespace="##other" processContents="strict" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>

<xsd:attributeGroup name="componentAttributes">
  <xsd:attribute name="class" type="xsd:string"/>
  <xsd:attribute name="scope" type="xsd:string"/>
  <xsd:attribute name="lazy-init" default="default" type="Tdefaultable-boolean"/>
  <xsd:attribute name="depends-on" type="xsd:string"/>
  <xsd:attribute name="init-method" type="xsd:string"/>
  <xsd:attribute name="destroy-method" type="xsd:string"/>
  <xsd:attribute name="factory-method" type="xsd:string"/>
  <xsd:attribute name="factory-component" type="xsd:string"/>
<xsd:element name="component">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="TidentifiedType">
        <xsd:group ref="componentElements"/>
        <xsd:attributeGroup ref="componentAttributes"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:group name="valueElement">
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="1">
      <xsd:element ref="component"/>
      <xsd:element ref="ref"/>
      <xsd:element ref="idref"/>
      <xsd:element ref="value"/>
      <xsd:element ref="null"/>
      <xsd:element ref="list"/>
      <xsd:element ref="set"/>
      <xsd:element ref="map"/>
      <xsd:element ref="array"/>
      <xsd:element ref="props"/>
      <xsd:element ref="ref-list"/>
      <xsd:element ref="ref-set"/>
      <xsd:element ref="reference"/>
      <xsd:element ref="service"/>
      <xsd:any namespace="##other" processContents="strict" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:group>

<xsd:group name="keyElement">
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="1">
      <xsd:element ref="component"/>
      <xsd:element ref="ref"/>
      <xsd:element ref="idref"/>
      <xsd:element ref="value"/>
      <xsd:element ref="list"/>
      <xsd:element ref="set"/>
      <xsd:element ref="map"/>
      <xsd:element ref="array"/>
      <xsd:element ref="props"/>
      <xsd:element ref="ref-list"/>
      <xsd:element ref="ref-set"/>
      <xsd:element ref="reference"/>
      <xsd:element ref="service"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:group>
</xsd:complexType>
</xsd:element>

<xsd:element name="list" type="TlistArrayType"/>
<xsd:element name="set" type="TlistArrayType"/>
<xsd:element name="map" type="TmapType"/>
<xsd:element name="array" type="TlistArrayType"/>
<xsd:element name="entry" type="TentryType"/>
<xsd:element name="props" type="TpropsType"/>
<xsd:element name="key">
  <xsd:complexType>
    <xsd:group ref="keyElement"/>
  </xsd:complexType>
</xsd:element>
</xsd:element>

<xsd:element name="prop">
  <xsd:complexType mixed="true">
    <xsd:attribute name="key" type="xsd:string" use="required"/>
    <xsd:attribute name="value" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>
</xsd:complexType>

<xsd:complexType name="TpropertyType">
  <xsd:sequence>
    <xsd:element ref="description" minOccurs="0"/>
    <xsd:attribute ref="valueElement"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="ref" type="xsd:string"/>
  <xsd:attribute name="value" type="xsd:string"/>
</xsd:complexType>

<!-- Collection Types -->

<!-- base collection type -->
<xsd:complexType name="TbaseCollectionType"/>

<!-- base type for collections that have (possibly) typed nested values -->
<xsd:complexType name="TtypedCollectionType">
  <xsd:complexContent>
    <xsd:extension base="TbaseCollectionType">
      <xsd:attribute name="value-type" type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- 'map' element type -->
<xsd:complexType name="TmapType">
  <xsd:complexContent>
    <xsd:extension base="TtypedCollectionType">
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="entry"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="TentryType">
  <xsd:sequence>
    <xsd:element ref="key" minOccurs="0"/>
    <xsd:group ref="valueElement"/>
  </xsd:sequence>
  <xsd:attribute name="key" type="xsd:string"/>
  <xsd:attribute name="key-ref" type="xsd:string"/>
  <xsd:attribute name="value" type="xsd:string"/>
  <xsd:attribute name="value-ref" type="xsd:string"/>
</xsd:complexType>

<!-- 'list' and 'set' collection type -->
<xsd:complexType name="TlistSetArrayType">
  <xsd:complexContent>
    <xsd:extension base="TtypedCollectionType">
      <xsd:group ref="valueElement" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- 'props' collection type -->
<xsd:complexType name="TpropsType">
  <xsd:complexContent>
    <xsd:extension base="TbaseCollectionType">
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="prop"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- simple internal types -->
<xsd:simpleType name="Tdefaultable-boolean">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="default"/>
    <xsd:enumeration value="true"/>
    <xsd:enumeration value="false"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:attributeGroup name="defaults">
  <xsd:attribute name="default-timeout" type="xsd:long" default="300000"/>
  <xsd:attribute name="default-availability" type="Tavailability" default="mandatory"/>
</xsd:attributeGroup>
<xsd:attributeGroup>

<!-- reference -->
<xsd:element name="reference" type="TsingleReference"/>

<xsd:complexType name="TReference">
  <xsd:complexContent>
    <xsd:extension base="TidentifiedType">
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="interfaces" type="TlistSetArrayType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="listener" type="Tlistener" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="interface" use="optional" type="xsd:token"/>
      <xsd:attribute name="filter" use="optional" type="xsd:string"/>
      <xsd:attribute name="component-name" type="xsd:string" use="optional"/>
      <xsd:attribute name="availability" use="optional" type="Tavailability" default="mandatory"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Tlistener">
  <xsd:choice>
    <xsd:element ref="ref"/>
    <xsd:element ref="component"/>
    <xsd:element ref="service"/>
  </xsd:choice>
  <xsd:sequence minOccurs="0" maxOccurs="1">
    <!-- nested component declaration -->
    <xsd:any namespace="##other" minOccurs="1" maxOccurs="1" processContents="skip"/>
  </xsd:sequence>
</xsd:complexType>

<!-- single reference -->
<xsd:complexType name="TsingleReference">
  <xsd:complexContent>
    <xsd:extension base="TReference">
      <xsd:attribute name="timeout" use="optional" type="xsd:long"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="Tavailability">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="mandatory"/>
    <xsd:enumeration value="optional"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:restriction>
  <xsd:simpleType>
    <!-- reference collections (set, list) -->
    <xsd:element name="ref-list" type="TreferenceCollection"/>
    <xsd:element name="ref-set" type="TreferenceCollection"/>
  </xsd:simpleType>
</xsd:restriction>

<xs:complexType name="TreferenceCollection">
  <xs:complexContent>
    <xs:extension base="Treference">
      <xs:sequence minOccurs="0" maxOccurs="1">
        <xs:element name="comparator" type="Tcomparator"/>
      </xs:sequence>
      <xs:attribute name="comparator-ref" type="xsd:string" use="optional"/>
      <xs:attribute name="member-type" type="TmemberType" use="optional"/>
      <xs:attribute name="ordering-basis" type="TorderingBasis" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:simpleType name="TmemberType">
  <xs:restriction base="xsd:token">
    <xs:enumeration value="service-instance"/>
    <xs:enumeration value="service-reference"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="Tcomparator">
  <xs:choice>
    <xs:element ref="ref"/>
    <xs:element ref="component"/>
    <xs:element ref="reference"/>
    <xs:element ref="service"/>
    <xs:sequence minOccurs="1" maxOccurs="1">
      <!-- nested component declaration -->
      <xs:any namespace="##other" minOccurs="1" maxOccurs="1" processContents="skip"/>
    </xs:sequence>
  </xs:choice>
</xs:complexType>

<xs:simpleType name="TorderingBasis">
  <xs:restriction base="xsd:token">
    <xs:enumeration value="service"/>
    <xs:enumeration value="service-reference"/>
  </xs:restriction>
</xs:simpleType>
<xsd:complexContent>
  <xsd:extension base="TidentifiedType">
    <xsd:choice>
      <xsd:group ref="serviceElements"/>
      <!-- nested component declaration -->
      <xsd:any namespace="##other" minOccurs="0" maxOccurs="1" processContents="skip"/>
    </xsd:choice>
    <xsd:attribute name="interface" type="xsd:token" use="optional"/>
    <xsd:attribute name="ref" type="xsd:string" use="optional"/>
    <xsd:attribute name="depends-on" type="xsd:string" use="optional"/>
    <xsd:attribute name="auto-export" type="TautoExportModes" default="disabled"/>
    <xsd:attribute name="ranking" type="xsd:int" default="0"/>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
</xsd:group>

<xsd:complexType name="TserviceRegistrationListener">
  <xsd:choice>
    <xsd:element ref="ref"/>
    <xsd:element ref="component"/>
    <xsd:element ref="reference"/>
    <xsd:element ref="service"/>
    <xsd:sequence minOccurs="0" maxOccurs="1">
      <!-- nested component declaration -->
      <xsd:any namespace="##other" minOccurs="1" maxOccurs="1" processContents="skip"/>
    </xsd:sequence>
  </xsd:choice>
  <!-- shortcut for component references -->
  <xsd:attribute name="ref" type="xsd:string" use="optional"/>
  <xsd:attribute name="registration-method" type="xsd:token" use="required"/>
  <xsd:attribute name="unregistration-method" type="xsd:token" use="required"/>
</xsd:complexType>

<xsd:simpleType name="TautoExportModes">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="disabled"/>
    <xsd:enumeration value="interfaces"/>
    <xsd:enumeration value="class-hierarchy"/>
    <xsd:enumeration value="all-classes"/>
  </xsd:restriction>
</xsd:simpleType>
5.24 'osgix' Schema

```xml
<xs:schema xmlns="http://www.osgi.org/xmlns/blueprint-compendium/v1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema"
    xmlns:osgi="http://www.osgi.org/xmlns/blueprint/v1.0.0"
    targetNamespace="http://www.osgi.org/xmlns/blueprint-compendium/v1.0.0"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    version="1.0.0">

  <xs:import namespace="http://www.osgi.org/xmlns/blueprint/v1.0.0"/>

  <!-- property placeholder -->

  <xs:element name="property-placeholder" type="TpropertyPlaceholder"/>

  <xs:complexType name="TpropertyPlaceholder">
    <xs:complexContent>
      <xs:extension base="osgi:TidentifiedType">
        <xs:sequence minOccurs="0" maxOccurs="1">
          <!-- nested properties declaration -->
          <xs:element name="default-properties" type="TdefaultProperties" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="persistent-id" type="xsd:string" use="required"/>
        <xs:attribute name="placeholder-prefix" type="xsd:string" use="optional" default=">${"}/
        <xs:attribute name="placeholder-suffix" type="xsd:string" use="optional" default="}"/>
        <xs:attribute name="defaults-ref" type="xsd:string" use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="TdefaultProperties">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="osgi:property"/>
    </xs:sequence>
  </xs:complexType>

  <!-- managed-properties -->

  <xs:element name="managed-properties" type="TmanagedProperties"/>

  <xs:complexType name="TmanagedProperties">
    <xs:attribute name="persistent-id" type="xsd:string" use="required"/>
    <xs:attribute name="update-strategy" type="TupdateStrategyType" use="optional"/>
    <xs:attribute name="update-method" type="xsd:string" use="optional"/>
  </xs:complexType>

  <xs:simpleType name="TupdateStrategyType">
    <!--...-->
  </xs:simpleType>
</xs:schema>
```
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="none"/>
  <xsd:enumeration value="component-managed"/>
  <xsd:enumeration value="container-managed"/>
</xsd:restriction>
</xsd:simpleType>

<!-- managed-service-factory -->

<xsd:element name="managed-service-factory" type="TmanagedServiceFactory"/>

<xsd:complexType name="TmanagedServiceFactory">
  <xsd:complexContent>
    <xsd:extension base="osgi:TidentifiedType">
      <xsd:sequence>
        <xsd:group ref="osgi:serviceElements"/>
        <xsd:element name="managed-component" type="TmanagedComponent" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="factory-pid" type="xsd:string" use="required"/>
      <xsd:attribute name="interface" type="xsd:token" use="optional"/>
      <xsd:attribute name="auto-export" type="osgi:TautoExportModes" default="disabled"/>
      <xsd:attribute name="ranking" type="xsd:int" default="0"/>
      <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="TmanagedComponent">
  <xsd:group ref="osgi:componentElements"/>
  <xsd:attribute name="class" type="xsd:string"/>
  <xsd:attribute name="init-method" type="xsd:string"/>
  <xsd:attribute name="destroy-method" type="xsd:string"/>
  <xsd:attribute name="factory-method" type="xsd:string"/>
  <xsd:attribute name="factory-component" type="xsd:string"/>
  <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>

<!-- cm-properties -->

<xsd:element name="cm-properties" type="TcmProperties"/>

<xsd:complexType name="TcmProperties">
  <xsd:attribute name="persistent-id" type="xsd:string" use="required"/>
  <xsd:attribute name="update" type="xsd:boolean" use="optional" default="false"/>
</xsd:complexType>

</xsd:schema>
6 Considered Alternatives

Todo: document considered alternatives for behavior of a mandatory reference that becomes unsatisfied.

6.1 Type Converters

Several alternative designs for registering type converters were considered:

6.1.1 Declaring type converters in a manifest header entry

The advantage of declaring type converters to be used when creating a module context for a bundle in the bundle’s manifest is that the container configuration is cleanly separated from the component instantiation.

The disadvantages were that the manifest entry looked a little out of place, and that it was not possible to have any control over the instantiation and configuration of the converters themselves.

6.1.2 Registering type converters as services in the service registry

Using the service registry would allow a standard whiteboard pattern for finding type converters. It initially seems attractive but has one serious disadvantage: it’s hard to create a compatible type space. Converters implement a simple interface that in its signature converts from Object to Object (no generics allowed). A type converter published in the service registry would advertise the “Converter” interface, and use a service property to indicate the name of the class it can convert to. But how do you guarantee that the version of the class the service uses is the same as the one that the bundle using the conversion service is bound too? This requires tricky walking through PackageAdmin before the converter can be safely used. Add to this the fact that sharing of type converter instances across bundles is of limited value, and the service registry approach becomes less attractive.

6.1.3 “Magic” component declarations

A design whereby simply declaring a component that implements the Converter interface (but not needing the nested type-converters element) was considered. The difficulty with this is that the container must discover all components implementing Converter before configuring any components, which forces a phased instantiation model on components and proves problematic with situations such as constructor args. In addition, walking the type hierarchy of a component instance to discover if it implements a certain interface can be problematic when supertypes are not visible to the bundle.

7 Security Considerations

7.1 Service Permissions

Service registration and lookup in the Blueprint Service is built upon the existing OSGi service infrastructure. This means that Service Permission applies regarding the ability to publish, find or bind services. A service element
declaring a component to be published as a service requires that the declaring bundle has ServicePermission[<provides>, REGISTER] for each provided interface specified for the service.

If a service reference is declared and does not specify optional cardinality, the reference cannot be satisfied unless the declaring bundle has ServicePermission[<interface>, GET] for the specified interface in the reference.

If the reference specifies optional cardinality but the declaring bundle does not have ServicePermission[<interface>, GET] for the specified interface in the reference, no service must be bound for this reference.

### 7.2 Required Admin Permission

A Blueprint Service implementation bundle requires AdminPermission[*,,CONTEXT] because it needs access to the Bundle’s BundleContext object with the Bundle.getBundleContext() method.

### 7.3 Using hasPermission

An implementation of the Blueprint Service should do all publishing, finding and binding of services for a Module Context using the BundleContext of the corresponding bundle. This means that normal stack-based permission checks will check the implementation bundle and not the module context’s bundle. Since the implementation is registering and getting services on behalf of a module context’s bundle, the implementation must call the Bundle.hasPermission method to validate that a module context’s bundle has the necessary permission to register or get a service.

---

### 8 Document Support

#### 8.1 References


#### 8.2 Author’s Address
8.3 Acronyms and Abbreviations

8.4 End of Document
Abstract
OSGi defines a rich command and control API for the core system. This API, however, is not well suited for remote management systems, nor does it provide a useful definition for layering a JMX management interface around. This specification defines a JMX compliant model for managing the OSGi framework and critical compendium services. This model consists of a set of interfaces and simple types which provide a more suitable management and monitoring API for remote management applications.
# 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

3 Problem Description....................................................................................................................5
  3.1 Integration with Existing Enterprise Management Systems..................................................5
  3.2 Integration with JMX Management Systems.........................................................................5
  3.3 High Fidelity Remote Control.................................................................................................5
  3.4 Security: Authentication and Access Control.........................................................................5
  3.5 Localization Issues..................................................................................................................6

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

5 Technical Solution.......................................................................................................................7
  5.1 Architectural Overview...........................................................................................................7
  5.2 The MBeanServer...................................................................................................................7
  5.3 Interfaces................................................................................................................................8
  5.4 JMX Object Names..................................................................................................................8
  5.5 CompositeData types.............................................................................................................8
    5.5.1 BundleBatchActionResult...............................................................................................8
    5.5.2 BundleBatchInstallResult...............................................................................................12
    5.5.3 OSGiAuthorization...........................................................................................................16
    5.5.4 OSGiBundle.......................................................................................................................18
    5.5.5 OSGiBundleEvent.............................................................................................................26
    5.5.6 OSGiGroup.........................................................................................................................28
    5.5.7 OSGiPackage......................................................................................................................31
    5.5.8 OSGiProperties..................................................................................................................34
    5.5.9 OSGiRole...........................................................................................................................40
    5.5.10 OSGiService...................................................................................................................42
    5.5.11 OSGiServiceEvent..........................................................................................................45
    5.5.12 OSGiUser.........................................................................................................................48
    5.5.13 Util..................................................................................................................................50
  5.6 Core Command and Control Interfaces....................................................................................55
    5.6.1 Interface FrameworkMBean.............................................................................................55
    5.6.2 Interface BundleStateMBean............................................................................................69
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 8.1.

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>0.1 draft</td>
<td>August 27, 2008</td>
<td>First draft of this RFC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hal Hildebrand, Oracle Corporation. <a href="mailto:hal.hildebrand@oracle.com">hal.hildebrand@oracle.com</a></td>
</tr>
<tr>
<td>0.2 draft</td>
<td>September 2, 2008</td>
<td>Incorporated comments from Richard Hall regarding naming, clarified use of symbolic name + version in arguments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hal Hildebrand, Oracle Corporation. <a href="mailto:hal.hildebrand@oracle.com">hal.hildebrand@oracle.com</a></td>
</tr>
<tr>
<td>0.3 draft</td>
<td>September 16, 2008</td>
<td>Fully fleshed out java doc of API, changed much of the fine grained access to deal in the currency of opaque identifiers (i.e. JMX Object Name strings) rather than (bundleSymbolicName;version) pairs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hal Hildebrand, Oracle Corporation. <a href="mailto:hal.hildebrand@oracle.com">hal.hildebrand@oracle.com</a></td>
</tr>
<tr>
<td>0.4 draft</td>
<td>September 23, 2008</td>
<td>Redesigned interfaces to eliminate individual Mbean representation for Bundles, Packages, and Services to allow the framework to effectively scale.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reformatted and transferred to Open Document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hal Hildebrand, Oracle Corporation. <a href="mailto:hal.hildebrand@oracle.com">hal.hildebrand@oracle.com</a></td>
</tr>
</tbody>
</table>
### 1 Introduction

The Java Management Extensions is the standard API specification for providing a management interface to J2SE and JEE applications. The JMX specification defines the design patterns, APIs, services and architecture for application and network management and monitoring in the Java programming language. The need to administer, monitor and manage a container is now 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. This document describes a proposal for an interface adaption of the existing OSGi framework which can be used to expose the OSGi framework manipulation API to any JMX compliant implementation. In addition, interfaces and system semantics for a monitoring system are proposed for exposing the underlying artifacts of the OSGi framework such as Services, Bundles, Packages, etc. 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.

It is important to note that JMX is not a remote communication standard. Rather, JMX is a specification which defines how arbitrary remote communication protocols and mechanisms can be adapted to interact with and control the underlying management apis exposed by compliant implementations.

The JMX architecture is composed of three levels:

- Instrumentation
- Agent
- Remote Management

At the instrumentation layer, the resources of the system are instrumented using Managed Beans which expose their management interfaces through a JMX agent for remote management and monitoring. 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 service for manipulating the MBeans, directly controls the resources and makes them available to remote management agents. 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 management applications outside of the Java process.

2 Application Domain

The primary domain addressed by this RFC is the management space of enterprise Java applications, although a solution to the requirements raised by the RFC should prove useful in other management domains.

3 Problem Description

Enterprise middleware includes infrastructure for manageability based on the Java Management Extensions framework. JMX is so widespread in the Java enterprise space that it has now become mandatory to expose management functions of middleware through standardized JMX APIs for the component. Consequently, if OSGi wishes to participate in modern enterprise management systems, exposing a JMX compliant management interface for the framework becomes a mandatory step to participating in the enterprise middleware space.

3.1 Integration with Existing Enterprise Management Systems
A significant number of Java enterprise middleware systems require JMX compliance in order to integrate into the management framework for the middleware platform. OSGi has an advantage over other containers in that the control over the base framework is both robust and well defined. Exposing this API through JMX allows these management systems to obtain high fidelity control in a framework independent fashion. This allows the OSGi container to integrate with the existing enterprise platform and participate fully in the solution space.

3.2 Integration with JMX Management Systems
JMX is the management standard for the Java standard edition, as well as the standard for managing the enterprise middleware space. JMX has turned out to be a popular standard in an endless number of application domains and providing a JMX compliant management and monitoring API would allow OSGi to seamlessly integrate with these management strategies as well.

3.3 High Fidelity Remote Control
It is quite common to require the control of an OSGi framework from outside the framework itself. The controlling system need not be “remote” in the traditional sense of the word, but could simply be in another class loading space within the same Java process. A carefully designed JMX API for the OSGi framework would allow systems to manipulate and control the OSGi container.
3.4 Security: Authentication and Access Control

Exposing any system remotely opens up a new, 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 interoperates with the Java Authentication and Authorization Service (JAAS) and Java 2 platform Standard Edition (J2SE) Security Architecture.

3.5 Localization Issues

Dealing with the intricacies and perils of localization is something that is expected by the customers of modern enterprise middleware platforms. JMX integrates well with existing Java mechanisms for dealing with the thorny issues involved in localization as well as defining it’s own sophisticated mechanisms for the localization issues created by having remote users to the framework, each of which may have different localization requirements.

4 Requirements

- The solution MUST be compliant to the Java Management Extensions specification.
- The solution MUST exploit Java 2 security if enabled by complying with the standard JMX security and authentication mechanisms.
- The solution MUST be remotely manageable through JMX compliant connectors and adapters.
- The solution SHOULD provide JMX models for all the core framework API, artifacts and services.
- The solution MUST define a uniform JMX object naming convention and format for uniquely identifying all JMX managed artifacts.
- The solution MUST provide access to all core OSGi framework APIs and services using primitive types and simple JRE classes such as Dictionary to allow the system to be used without requiring additional classes on the JMX client.
- The solution MAY define a handful of additional types for optional APIs which deal with complicated, aggregated framework state such as Bundles, Services and Packages.
- The solution SHOULD define a JMX integration of critical compendium services such as the configuration administration, logging, event administration and preferences services.
- The solution SHOULD allow the dynamic update of the JMX environment’s view of the OSGi framework state through the use of OSGi and JMX provided callbacks and listeners.
- The solution SHOULD be as simple as is possible, making use of no JMX specific artifacts to accomplish the goal, so that non JMX management frameworks may reuse of the resulting framework without pulling in JMX.
- The solution SHOULD NOT define any new models for manipulating the OSGi framework other than those already present in the existing framework APIs.
- The solution SHOULD NOT define a generic framework for JMX integration of arbitrary OSGi services. The goal is to manage the OSGi framework through JMX, not provide a generic mechanism that be used to expose management of arbitrary OSGi services through JMX.
5 Technical Solution

5.1 Architectural Overview

A set of Java interfaces defining what are known in JMX as Standard MBeans are created which closely follow the underlying OSGi command and control API. These interfaces define the manageable attributes and operations exposed through the JMX API. Standard MBeans were chosen following JMX best practices. Specifying these APIs using Standard MBeans allows them to be documented using the familiar Javadoc tool, and it allows client code to interact with them straightforwardly via proxies, using MBeanServerInvocationHandler. Contrast the code without a proxy:

```java
MBeanServer mbs = ...;
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"});
}
```

with the code that uses a proxy:

```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. This is much easier to write and read, and much less error-prone, than accessing the MBeanServer method directly.

All managed objects in JMX are referenced via the JMX Object Names. JMX Object Names are strings which can be resolved within the context of a JMX MBeanServer in order to invoke operations on the MBean referred to by the name. In this RFC, all interfaces are specified to return opaque Strings rather than actual JMX 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.

5.2 The MBeanServer

The construction, maintenance and lifecycle of the MBeanServer which will host the MBeans defined in this specification is intentionally left undefined. It is left undefined as the MBeanServer is invariably tied to the particular application that is responsible for it. For example, the MBeanServer may exist outside the OSGi framework that the MBeans are managing. Or there may be multiple MBean servers which contain the MBeans defined in this specification. The introduction of nested frameworks, such as those defined in RFC 138, may have their management MBeans hosted in the MBeanServer which hosts the MBeans for the outermost OSGi container. Alternatively, these MBeans may be hosted, for example, in an application server's MBeanServer which is embedding mulitple OSGi containers.
5.3 Interfaces

The API is divided into 2 interface sets. The first defines the management interface for the OSGi core command and control API. The second defines the management interface for essential compendium service APIs.

Note on JMX interface naming conventions wrt Standard MBeans: When using JMX Standard MBeans, the standard states that you have a Class which represents the actual implementation and the interface which is used by the JMX infrastructure. Standard MBeans work off of a naming convention where the interface the implementation implements is named <BaseClass>.<MBean>. Thus, the JMX class which implements the BundleStateMBean is named BundleState.

5.4 JMX Object Names

This RFC defines 8 JMX MBeans. These MBeans are bound to JMX ObjectNames within a MBeanServer. The standard ObjectNames for these MBeans are:

<table>
<thead>
<tr>
<th>MBean</th>
<th>JMX ObjectName</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameworkMBean</td>
<td>osgi.core::type=framework</td>
</tr>
<tr>
<td>BundleStateMBean</td>
<td>osgi.core::type=bundleState</td>
</tr>
<tr>
<td>PackageStateMBean</td>
<td>osgi.core::type=packageState</td>
</tr>
<tr>
<td>ServiceStateMBean</td>
<td>osgi.core::type=serviceState</td>
</tr>
<tr>
<td>ConfigAdminManagerMBean</td>
<td>osgi.compendium::service=configAdminManager</td>
</tr>
<tr>
<td>PermissionManagerMBean</td>
<td>osgi.compendium::service=permissionAdmin</td>
</tr>
<tr>
<td>ProvisioningMBean</td>
<td>osgi.compendium::service=provisioningService</td>
</tr>
<tr>
<td>UserManagerMBean</td>
<td>osgi.compendium::service=userAdmin</td>
</tr>
</tbody>
</table>

5.5 CompositeData types

Some of the operations in this API return JMX open data types represented by CompositeData. These composite data types are facilitated by a set of codec classes which serve to document the structure of the Composite data as well as provide a convenient mechanism to represent the CompositeData type within Java as well as the conversion to and from the composite data type.

5.5.1 BundleBatchActionResult

org.osgi.jmx.codec Class BundleBatchActionResult

Object

   org.osgi.jmx.codec.BundleBatchActionResult

public class BundleBatchActionResult
extends Object

This class represents the CODEC for the resulting composite data from the batch operations on the bundles in the FrameworkMBean. It serves as both the documentation of the type structure and as the codification of the mechanism to convert to/from the CompositeData.
The structure of the composite data is:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
</tr>
<tr>
<td>Completed</td>
<td>Array of long</td>
</tr>
<tr>
<td>BundleInError</td>
<td>long</td>
</tr>
<tr>
<td>Remaining</td>
<td>Array of long</td>
</tr>
</tbody>
</table>

**Field Summary**

```
static CompositeType RESULT
The CompositeType which represents the result of batch operations on the FrameworkMBean
```

**Constructor Summary**

```
BundleBatchActionResult()
Construct a result signifying the successful completion of the batch operation.

BundleBatchActionResult(CompositeData compositeData)
Construct a result representing the contents of the supplied CompositeData returned from a batch operation.

BundleBatchActionResult(String errorMessage, long[] completed, long bundleInError, long[] remaining)
Construct a result indicating the failure of a batch operation.
```

**Method Summary**

```
CompositeData asCompositeData()
Answer the receiver encoded as CompositeData

long getBundleInError()
Answer the bundle identifier which indicates the bundle that produced an error during the batch operation.

long[] getCompleted()
If the operation failed, answer the list of bundle identifiers that successfully completed the batch operation.

String getErrorMessage()
Answer the error message indicating the error that occurred during the batch operation or null, if the operation was a success.

long[] getRemaining()
If the operation was unsuccessful, answer the list of bundle identifiers of the bundles that were not processed during the batch operation.
```
boolean isSuccess()  
Answer true if the batch operation was successful, false otherwise.

Methods inherited from class Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail
RESULT

public static final CompositeType RESULT
The CompositeType which represents the result of batch operations on the FrameworkMBean

Constructor Detail
BundleBatchActionResult

public BundleBatchActionResult()
Construct a result signifying the successful completion of the batch operation.

BundleBatchActionResult

public BundleBatchActionResult(CompositeData compositeData)
Construct a result representing the contents of the supplied CompositeData returned from a batch operation.

Parameters:
compositeData -- the CompositeData representing the result of a batch operation.

BundleBatchActionResult

public BundleBatchActionResult(String errorMessage,  
long[] completed,  
long bundleInError,  
long[] remaining)
Construct a result indictating the failure of a batch operation.

Parameters:
errorMessage -- the message indicating the error
completed -- the list of bundle identifiers indicating bundles that have successfully completed the batch operation
bundleInError -- the identifier of the bundle which produced the error
remaining -- the list of bundle identifiers which remain unprocessed

Method Detail

asCompositeData

```java
public CompositeData asCompositeData()
Answer the receiver encoded as CompositeData

Returns:
the CompositeData encoding of the receiver.
```

getBundleInError

```java
public long getBundleInError()
Answer the bundle identifier which indicates the bundle that produced an error during the batch operation.

Returns:
the bundle identifier of the bundle in error, or -1L if no error occurred
```

getCompleted

```java
public long[] getCompleted()
If the operation failed, answer the list of bundle identifiers that successfully completed the batch operation. If the operation was successful, then the list is null;

Returns:
the list of bundle identifiers or null if the operation was successful
```

getErrorMessage

```java
public String getErrorMessage()
Answer the error message indicating the error that occurred during the batch operation or null, if the operation was a success.

Returns:
the String error message
```

getRemaining

```java
public long[] getRemaining()
```
If the operation was unsuccessful, answer the list of bundle identifiers of the bundles that were not processed during the batch operation. If the operation was a success, then answer null.

**Returns:**
the remaining bundle identifiers or null if the operation was a success

**isSuccess**

```java
public boolean isSuccess()
```

Answer true if the batch operation was successful, false otherwise.

**Returns:**
the success of the batch operation

### 5.5.2 BundleBatchInstallResult

**org.osgi.jmx.codec**

**Class BundleBatchInstallResult**

**Object**

```java
org.osgi.jmx.codec.BundleBatchInstallResult
```

**public class BundleBatchInstallResult**

**extends Object**

This class represents the CODEC for the resulting composite data from the batch install operations on the bundles in the FrameworkMBean. It serves as both the documentation of the type structure and as the codification of the mechanism to convert to/from the CompositeData.

The structure of the composite data is:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
</tr>
<tr>
<td>Completed</td>
<td>Array of long</td>
</tr>
<tr>
<td>BundleInError</td>
<td>String</td>
</tr>
<tr>
<td>Remaining</td>
<td>Array of String</td>
</tr>
</tbody>
</table>

**Field Summary**

<table>
<thead>
<tr>
<th>static CompositeType</th>
<th>BATCH_RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The CompositeType which represents the result of batch install operations on the FrameworkMBean</td>
</tr>
</tbody>
</table>
Constructor Summary

**BundleBatchInstallResult**((CompositeData compositeData))
Construct a result representing the contents of the supplied CompositeData returned from a batch operation.

**BundleBatchInstallResult**((long[] completed))
Construct a result signifying the successful completion of the batch operation.

**BundleBatchInstallResult**((String errorMessage, long[] completed, String bundleInError, String[] remaining))
Construct a result indicating the failure of a batch operation.

Method Summary

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Method Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompositeData</td>
<td><code>asCompositeData()</code></td>
<td>Answer the receiver encoded as CompositeData</td>
</tr>
<tr>
<td>String</td>
<td><code>getBundleInError()</code></td>
<td>Answer the bundle location which indicates the bundle that produced an error during the batch operation.</td>
</tr>
<tr>
<td>long[]</td>
<td><code>getCompleted()</code></td>
<td>Answer the list of bundle identifiers that successfully completed the batch operation.</td>
</tr>
<tr>
<td>String</td>
<td><code>getErrorMessage()</code></td>
<td>Answer the error message indicating the error that occurred during the batch operation or null if the operation was successful.</td>
</tr>
<tr>
<td>String[]</td>
<td><code>getRemaining()</code></td>
<td>Answer the list of locations of the bundles that were not processed during the batch operation, or null if the operation was successful.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>isSuccess()</code></td>
<td>Answer true if the batch operation was successful, false otherwise.</td>
</tr>
</tbody>
</table>

Methods inherited from class Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

**BATCH_RESULT**

public static final CompositeType **BATCH_RESULT**
The CompositeType which represents the result of batch install operations on the FrameworkMBean.
**Constructor Detail**

**BundleBatchInstallResult**

```java
public BundleBatchInstallResult(CompositeData compositeData)
```

Construct a result representing the contents of the supplied CompositeData returned from a batch operation.

**Parameters:**
compositeData -- the CompositeData representing the result of a batch operation.

**BundleBatchInstallResult**

```java
public BundleBatchInstallResult(long[] completed)
```

Construct a result signifying the successful completion of the batch operation.

**Parameters:**
completed -- the resulting bundle identifiers of the installed bundles

**BundleBatchInstallResult**

```java
public BundleBatchInstallResult(String errorMessage,
                                 long[] completed,
                                 String bundleInError,
                                 String[] remaining)
```

Construct a result indicating the failure of a batch operation.

**Parameters:**
errorMessage -- the message indicating the error
completed -- the list of bundle identifiers indicating bundles that have successfully completed the batch operation
bundleInError -- the identifier of the bundle which produced the error
remaining -- the list of bundle identifiers which remain unprocessed

**Method Detail**

**asCompositeData**

```java
public CompositeData asCompositeData()
```

Answer the receiver encoded as CompositeData

**Returns:**
the CompositeData encoding of the receiver.

**getBundleInError**
public String getBundleInError()
Answer the bundle location which indicates the bundle that produced an error during the batch operation.

 Returns:
the bundle location of the bundle in error, or null if no error occurred

getCompleted

public long[] getCompleted()
Answer the list of bundle identifiers that successfully completed the batch operation. If the operation was unsuccessful, this will be a partial list. If this operation was successful, this will be the full list of bundle ids. This list corresponds one to one with the supplied list of bundle locations provided to the batch install operations.

 Returns:
the list of identifiers of the bundles that successfully installed

getAddressMessage

public String getErrorMessage()
Answer the error message indicating the error that occurred during the batch operation or null if the operation was successful.

 Returns:
the String error message if the operation was unsuccessful, or null if the operation was successful

getRemaining

public String[] getRemaining()
Answer the list of locations of the bundles that were not processed during the batch operation, or null if the operation was successful.

 Returns:
the remaining bundle locations if the operation was successful, or null if the operation was unsuccessful.

isSuccess

public boolean isSuccess()
Answer true if the batch operation was successful, false otherwise.

 Returns:
the success of the batch operation
5.5.3 OSGiAuthorization

Class OSGiAuthorization

Object

```
java
public class OSGiAuthorization
extends Object
```

### Field Summary

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static CompositeType</td>
<td>AUTHORIZATION</td>
<td></td>
</tr>
<tr>
<td>protected String</td>
<td>name</td>
<td></td>
</tr>
<tr>
<td>protected String[]</td>
<td>roles</td>
<td></td>
</tr>
</tbody>
</table>

### Constructor Summary

- `OSGiAuthorization(Authorization authorization)`
- `OSGiAuthorization(CompositeData data)`
- `OSGiAuthorization(String name, String[] roles)`

### Method Summary

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompositeData</td>
<td>asCompositeData()</td>
</tr>
<tr>
<td>String</td>
<td>getName()</td>
</tr>
<tr>
<td>String[]</td>
<td>getRoles()</td>
</tr>
</tbody>
</table>

Methods inherited from class Object:
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Field Detail

**name**

protected String **name**

**roles**
protected String[] roles

AUTHORIZATION

public static final CompositeType AUTHORIZATION

Constructor Detail
OSGiAuthorization

public OSGiAuthorization(CompositeData data)

OSGiAuthorization

public OSGiAuthorization(Authorization authorization)

OSGiAuthorization

public OSGiAuthorization(String name, String[] roles)

Method Detail
asCompositeData

public CompositeData asCompositeData() throws OpenDataException

Throws:
OpenDataException

getName

public String getName()

Returns:
the name

getRoles

public String[] getRoles()

Returns:
the roles
5.5.4 OSGiBundle

Object

```
org.osgi.jmx.codec OSGiBundle
```

define class

```
public class OSGiBundle
extends Object
```

This class represents the CODEC for the composite data representing a single OSGi Bundle.

It serves as both the documentation of the type structure and as the codification of the mechanism to convert to/from the CompositeData.

The structure of the composite data is:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>String</td>
</tr>
<tr>
<td>Identifier</td>
<td>long</td>
</tr>
<tr>
<td>SymbolicName</td>
<td>String</td>
</tr>
<tr>
<td>StartLevel</td>
<td>int</td>
</tr>
<tr>
<td>State</td>
<td>String</td>
</tr>
<tr>
<td>LastModified</td>
<td>long</td>
</tr>
<tr>
<td>PersistentlyStarted</td>
<td>boolean</td>
</tr>
<tr>
<td>RemovalPending</td>
<td>boolean</td>
</tr>
<tr>
<td>Required</td>
<td>boolean</td>
</tr>
<tr>
<td>Fragment</td>
<td>boolean</td>
</tr>
<tr>
<td>RegisteredServices</td>
<td>Array of long</td>
</tr>
<tr>
<td>ServicesInUse</td>
<td>Array of long</td>
</tr>
<tr>
<td>Headers</td>
<td>TabularData of Key/Value String pairs</td>
</tr>
<tr>
<td>ExportedPackages</td>
<td>Array of String</td>
</tr>
<tr>
<td>ImportedPackages</td>
<td>Array of String</td>
</tr>
<tr>
<td>Fragments</td>
<td>Array of long</td>
</tr>
<tr>
<td>Hosts</td>
<td>Array of long</td>
</tr>
<tr>
<td>RequiredBundles</td>
<td>Array of long</td>
</tr>
<tr>
<td>RequiringBundles</td>
<td>Array of long</td>
</tr>
</tbody>
</table>

**Field Summary**

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static CompositeType</td>
<td><strong>BUNDLE</strong></td>
</tr>
<tr>
<td>static CompositeType</td>
<td><strong>BUNDLE_HEADER</strong></td>
</tr>
<tr>
<td>static TabularType</td>
<td><strong>BUNDLE_HEADER_TABLE</strong></td>
</tr>
</tbody>
</table>

Copyright © Oracle Corporation 2009 All Rights Reserved
The TabularType which represents the map of bundle headers

static TabularType **BUNDLE TABLE**

The TabularType which represents a list of bundles

### Constructor Summary

**OSGiBundle**(BundleContext bc, PackageAdmin admin, StartLevel sl, Bundle b)

Construct an OSGiBundle representation

**OSGiBundle**(CompositeData data)

Construct an OSGiBundle from the encoded CompositeData

**OSGiBundle**(String location, long identifier, String symbolicName, int startLevel, String state, long lastModified, boolean persistentlyStarted, boolean removalPending, boolean required, boolean fragment, long[] registeredServices, long[] servicesInUse, java.util.Map<String,String> headers, String[] exportedPackages, String[] importedPackages, long[] fragments, long[] hosts, long[] requiredBundles, long[] requiringBundles)

Construct and OSGiBundle

### Method Summary

**CompositeData**

**asCompositeData**()

Answer the receiver encoded as CompositeData

**String[]**

**getExportedPackages**()

**long[]**

**getFragments**()

**java.util.Map<String,String>**

**getHeaders**()

**long[]**

**getHosts**()

**long**

**getIdentifier**()

**String[]**

**getImportedPackages**()

**long**

**getLastModified**()

**String**

**getLocation**()

**long[]**

**getRegisteredServices**()

**long[]**

**getRequiredBundles**()

**long[]**

**getRequiringBundles**()
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>long[]</code> <code>getServicesInUse()</code></td>
<td></td>
</tr>
<tr>
<td><code>int</code> <code>getStartLevel()</code></td>
<td></td>
</tr>
<tr>
<td><code>String</code> <code>getState()</code></td>
<td></td>
</tr>
<tr>
<td><code>String</code> <code>getSymbolicName()</code></td>
<td></td>
</tr>
<tr>
<td><code>static TabularData</code> <code>headerTable(Bundle b)</code></td>
<td>Answer the TabularData representing the list of bundle headers for a bundle</td>
</tr>
<tr>
<td><code>static TabularData</code> <code>headerTable(java.util.Map&lt;String, String&gt; headers)</code></td>
<td></td>
</tr>
<tr>
<td><code>boolean</code> <code>isFragment()</code></td>
<td></td>
</tr>
<tr>
<td><code>boolean</code> <code>isPersistentlyStarted()</code></td>
<td></td>
</tr>
<tr>
<td><code>boolean</code> <code>isRemovalPending()</code></td>
<td></td>
</tr>
<tr>
<td><code>boolean</code> <code>isRequired()</code></td>
<td></td>
</tr>
<tr>
<td><code>static TabularData</code> <code>tableFrom(java.util.ArrayList&lt;OSGiBundle&gt; bundles)</code></td>
<td>Answer the TabularData representing the list of OSGiBundle state</td>
</tr>
</tbody>
</table>

Methods inherited from class `Object`:
- `clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`.

### Field Detail

**BUNDLE_HEADER**

public static final CompositeType `BUNDLE_HEADER`

The CompositeType which represents a key/value header pair.

**BUNDLE_HEADER_TABLE**

public static final TabularType `BUNDLE_HEADER_TABLE`

The TabularType which represents the map of bundle headers.
**BUNDLE**

```java
public static final CompositeType BUNDLE
The CompositeType which represents a single OSGi bundle
```

**BUNDLE_TABLE**

```java
public static final TabularType BUNDLE_TABLE
The TabularType which represents a list of bundles
```

---

**Constructor Detail**

**OSGiBundle**

```java
public OSGiBundle(CompositeData data)
Construct an OSGiBundle from the encoded CompositeData
```

**Parameters:**

data -- the encoded representation of the bundle

---

**OSGiBundle**

```java
public OSGiBundle(BundleContext bc,
                  PackageAdmin admin,
                  StartLevel sl,
                  Bundle b)
Construct an OSGiBundle representation
```

**Parameters:**

bc - - the BundleContext to be used.
admin - - the PackageAdmin service
sl - - the StartLevel service
b - - the Bundle to represent

---

**OSGiBundle**

```java
public OSGiBundle(String location,
                   long identifier,
                   String symbolicName,
                   int startLevel,
```
String state,
long lastModified,
boolean persistentlyStarted,
boolean removalPending,
boolean required,
boolean fragment,
long[] registeredServices,
long[] servicesInUse,
java.util.Map<String,String> headers,
String[] exportedPackages,
String[] importedPackages,
long[] fragments,
long[] hosts,
long[] requiredBundles,
long[] requiringBundles)

Construct and OSGiBundle

Method Detail

public static TabularData
tableFrom(java.util.ArrayList<OSGiBundle> bundles)
Answer the TabularData representing the list of OSGiBundle state

Parameters:
bundles -- the list of bundles to represent

Returns:
the Tabular data which represents the list of bundles
headerTable

public static TabularData headerTable(Bundle b)
Answer the TabularData representing the list of bundle headers for a bundle

Parameters:
b -

Returns:
the bundle headers

headerTable

public static TabularData headerTable(java.util.Map<String,String> headers)

asCompositeData

public CompositeData asCompositeData()
Answer the receiver encoded as CompositeData

Returns:
the CompositeData encoding of the receiver.

getExportedPackages

public String[] getExportedPackages()
Returns:
The list of exported packages by this bundle, in the form of ;

getFragments

public long[] getFragments()
Returns:
the list of identifiers of the bundle fragments which use this bundle as a host

getHeaders

public java.util.Map<String,String> getHeaders()
Returns:
the map of headers for this bundle

getHosts

public long[] getHosts()
Returns:
list of identifiers of the bundles which host this fragment

`getIdentifier`

```java
public long getIdentifier()
```

Returns:
the identifier of this bundle

`getImportedPackages`

```java
public String[] getImportedPackages()
```

Returns:
The list of imported packages by this bundle, in the form of ;

`getLastModified`

```java
public long getLastModified()
```

Returns:
the last modified time of this bundle

`getLocation`

```java
public String getLocation()
```

Returns:
the name of this bundle

`getRegisteredServices`

```java
public long[] getRegisteredServices()
```

Returns:
the list of identifiers of the services registered by this bundle

`getRequiredBundles`

```java
public long[] getRequiredBundles()
```

Returns:
the list of identifiers of bundles required by this bundle

`getRequiringBundles`

```java
public long[] getRequiringBundles()
```

Returns:
the list of identifiers of bundles which require this bundle

`getServicesInUse`
public long[]} getServicesInUse()

**Returns:**
the list of identifiers of services in use by this bundle

**getStartLevel**

public int getStartLevel()

**Returns:**
the start level of this bundle

**getState**

public String getState()

**Returns:**
the state of this bundle

**getSymbolicName**

public String getSymbolicName()

**Returns:**
the symbolic name of this bundle

**isFragment**

public boolean isFragment()

**Returns:**
true if this bundle represents a fragment

**isPersistentlyStarted**

public boolean isPersistentlyStarted()

**Returns:**
true if this bundle is persistently started

**isRemovalPending**

public boolean isRemovalPending()

**Returns:**
true if this bundle is pending removal

**isRequired**

public boolean isRequired()

**Returns:**
true if this bundle is required
5.5.5 OSGiBundleEvent

org.osgi.jmx.codec Class OSGiBundleEvent

public class OSGiBundleEvent
    extends Object

Author:
Hal Hildebrand Date: Nov 24, 2008 Time: 2:22:34 PM
This class represents the CODEC for the composite data representing a OSGi BundleEvent

It serves as both the documentation of the type structure and as the codification of the mechanism to convert to/from the CompositeData.

The structure of the composite data is:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>long</td>
</tr>
<tr>
<td>location</td>
<td>String</td>
</tr>
<tr>
<td>SymbolicName</td>
<td>String</td>
</tr>
<tr>
<td>EventType</td>
<td>int</td>
</tr>
</tbody>
</table>

Field Summary

static CompositeType BUNDLE_EVENT
    The CompositeType representation of the event

Constructor Summary

OSGiBundleEvent(BundleEvent event)
    Construct an OSGiBundleEvent from the supplied BundleEvent

OSGiBundleEvent(CompositeData data)
    Construct an OSGiBundleEvent from the CompositeData representing the event

OSGiBundleEvent(long bundleId, String location, String symbolicName, int eventType)
    Construct the OSGiBundleEvent

Method Summary

CompositeData asCompositeData()
    Answer the receiver encoded as CompositeData

long getBundleId()

int getEventType()
Methods inherited from class Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail
BUNDLE_EVENT

public static final CompositeType BUNDLE_EVENT
The CompositeType representation of the event

Constructor Detail
OSGiBundleEvent

public OSGiBundleEvent(BundleEvent event)
Construct an OSGiBundleEvent from the supplied BundleEvent

Parameters:
event -- the event to represent

OSGiBundleEvent

public OSGiBundleEvent(CompositeData data)
Construct an OSGiBundleEvent from the CompositeData representing the event

Parameters:
data -- the CompositeData representing the event.

OSGiBundleEvent

public OSGiBundleEvent(long bundleId, String location, String symbolicName, int eventType)
Construct the OSGiBundleEvent
Parameters:
bundleId -
location -
symbolicName -
eventType -

Method Detail

asCompositeData

public CompositeData asCompositeData()
Answer the receiver encoded as CompositeData

Returns:
the CompositeData encoding of the receiver.

getBundleId

public long getBundleId()
Returns:
the identifier of the bundle for this event

gETFEventTypet

public int getEventType()
Returns:
the type of the event

gETLocation

public String getLocation()
Returns:
the location of the bundle for this event

gETFSymmetricName

public String getSymbolicName()
Returns:
the symbolic name of the bundle for this event

5.5.6 OSGiGroup

org.osgi.jmx.codec Class OSGiGroup

Object
org.osgi.jmx.codec.OSGiGroup

public class OSGiGroup
extends Object

### Field Summary

<table>
<thead>
<tr>
<th>CompositeType</th>
<th>GROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected String[]</td>
<td>members</td>
</tr>
<tr>
<td>protected String[]</td>
<td>requiredMembers</td>
</tr>
<tr>
<td>protected OSGiUser</td>
<td>user</td>
</tr>
</tbody>
</table>

### Constructor Summary

- **OSGiGroup**(CompositeData data)
- **OSGiGroup**(Group group)

### Method Summary

<table>
<thead>
<tr>
<th>CompositeData</th>
<th>asCompositeData()</th>
</tr>
</thead>
<tbody>
<tr>
<td>String[]</td>
<td>getMembers()</td>
</tr>
<tr>
<td>String[]</td>
<td>getRequiredMembers()</td>
</tr>
<tr>
<td>OSGiUser</td>
<td>getUser()</td>
</tr>
</tbody>
</table>

Methods inherited from class Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Field Detail

**user**

protected OSGiUser user

**members**

protected String[] members
requiredMembers

protected String[] requiredMembers

GROUP

public final CompositeType GROUP

Constructor Detail

OSGiGroup

public OSGiGroup(Group group)

OSGiGroup

public OSGiGroup(CompositeData data)

Method Detail

asCompositeData

public CompositeData asCompositeData() throws OpenDataException

Throws:
OpenDataException

getUser

public OSGiUser getUser()

Returns:
the user

getMembers

public String[] getMembers()

Returns:
the members

getRequiredMembers

public String[] getRequiredMembers()

Returns:
the requiredMembers
5.5.7 OSGiPackage

org.osgi.jmx.codec Class OSGiPackage

public class OSGiPackage
extends Object

This class represents the CODEC for the composite data representing an OSGi ExportedPackage.

It serves as both the documentation of the type structure and as the codification of the mechanism to convert to/from the CompositeData.

The structure of the composite data is:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
</tr>
<tr>
<td>Version</td>
<td>String</td>
</tr>
<tr>
<td>PendingRemoval</td>
<td>boolean</td>
</tr>
<tr>
<td>BundleIdentifier</td>
<td>long</td>
</tr>
<tr>
<td>ImportingBundles</td>
<td>Array of long</td>
</tr>
</tbody>
</table>

Field Summary

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static CompositeType PACKAGE</td>
<td>The CompositeType representation of the package</td>
</tr>
<tr>
<td>static TabularType PACKAGE_TABLE</td>
<td>The TabularType representation of a list of packages</td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSGiPackage(CompositeData data)</td>
<td>Construct an OSGiPackage from the encoded CompositeData</td>
</tr>
<tr>
<td>OSGiPackage(ExportedPackage pkg)</td>
<td>Construct an OSGiPackage from the ExportedPackage</td>
</tr>
<tr>
<td>OSGiPackage(String name, String version, boolean removalPending, long exportingBundle, long[] importingBundles)</td>
<td>Construct and OSGiPackage from the supplied data</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompositeData asCompositeData()</td>
<td>Answer the receiver encoded as CompositeData</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>long getExportingBundle()</td>
<td></td>
</tr>
<tr>
<td>long[] getImportingBundles()</td>
<td></td>
</tr>
<tr>
<td>String getName()</td>
<td></td>
</tr>
<tr>
<td>String getVersion()</td>
<td></td>
</tr>
<tr>
<td>boolean isRemovalPending()</td>
<td></td>
</tr>
<tr>
<td>static TabularData tableFrom(java.util.ArrayList&lt;OSGiPackage&gt; packages)</td>
<td></td>
</tr>
</tbody>
</table>

**Methods inherited from class Object**
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Field Detail**

**PACKAGE**

public static final CompositeType PACKAGE
The CompositeType representation of the package

**PACKAGE_TABLE**

public static final TabularType PACKAGE_TABLE
The TabularType representation of a list of packages

**Constructor Detail**

OSGiPackage

public OSGiPackage(CompositeData data)
Construct an OSGiPackage from the encoded CompositeData

**Parameters:**
data - - the CompositeData encoding the OSGiPackage

Copyright © Oracle Corporation 2009 All Rights Reserved
public OSGiPackage(ExportedPackage pkg)
Construct an OSGiPackage from the ExportedPackage

Parameters:
pkg -- the ExportedPackage

OSGiPackage

public OSGiPackage(String name,
String version,
boolean removalPending,
long exportingBundle,
long[] importingBundles)
Construct an OSGiPackage from the supplied data

Parameters:
name -
version -
removalPending -
exportingBundle -
importingBundles -

Method Detail

tableFrom

public static TabularData
    tableFrom(java.util.ArrayList<OSGiPackage> packages)

asCompositeData

public CompositeData asCompositeData()
Answer the receiver encoded as CompositeData

Returns:
the CompositeData encoding of the receiver.

getExportingBundle

public long getExportingBundle()
Returns:
the identifier of the exporting bundle

getImportingBundles

public long[] getImportingBundles()
Returns:
the list of identifiers of the bundles importing this package

getName

public String getName()

Returns:
the name of the package

getVersion

public String getVersion()

Returns:
the version of the package

isRemovalPending

public boolean isRemovalPending()

Returns:
true if the package is pending removal

5.5.8 OSGiProperties

org.osgi.jmx.codec Class OSGiProperties

Object

org.osgi.jmx.codec.OSGiProperties

public class OSGiProperties
extends Object

This class serves as both the documentation of the type structure and as the codification of the mechanism to convert to/from the TabularData.

This class represents the CODEC for property dictionaries. As JMX is a rather primitive system and is not intended to be a generic RMI type system, the set of types that can be transfered between the management agent and the managed OSGi container is limited to simple types, arrays of simple types and vectors of simple types. This enforcement is strict and no attempt is made to create a yet another generic serialization mechanism for transferring property values outside of these types.

The syntax for the type indicator

type ::= scalar | vector | array
scalar ::= String | Integer | Long | Float | Double | Byte | Short | Character | Boolean | BigDecimal | BigInteger
primitive ::= int | long | float | double | byte | short | char | boolean
array ::=     <Array of primitive> | <Array of scalar>
vector ::=    Vector of scalar

The values for Arrays and Vectors are separated by ",".
The structure of the composite data for a row in the table is:

<table>
<thead>
<tr>
<th>Key</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>String</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

Field Summary

| static String | KEY |
| protected static java.util.Set<String> | PRIMITIVE_TYPES |
| static String[] | PROPERTIES |
| static CompositeType | PROPERTY |
| static TabularType | PROPERTY_TABLE |
| protected static java.util.Set<String> | SCALAR_TYPES |
| static String | TYPE |
| static String | VALUE |

Constructor Summary

**OSGiProperties()**

Method Summary

<p>| protected static Object[] | createScalarArray(String type, int size) |
| static CompositeData | encode(String key, Object value) |
| protected static CompositeData | encodeArray(String key, Object value, Class&lt;?&gt; componentClazz) |
| protected static CompositeData | encodeVector(String key, |</p>
<table>
<thead>
<tr>
<th>static Object</th>
<th>java.util.Vector value</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected static Object</td>
<td>parse(String value, String type)</td>
</tr>
<tr>
<td>protected static Object</td>
<td>parseArray(String value, java.util.StringTokenizer tokens)</td>
</tr>
<tr>
<td>protected static Object</td>
<td>parsePrimitiveArray(String value, String type)</td>
</tr>
<tr>
<td>protected static Object</td>
<td>parseScalar(String value, String type)</td>
</tr>
<tr>
<td>protected static Object</td>
<td>parseScalarArray(String value, String type)</td>
</tr>
<tr>
<td>protected static Object</td>
<td>parseVector(String value, java.util.StringTokenizer tokens)</td>
</tr>
<tr>
<td>static java.util.Hashtable&lt;String, Object&gt;</td>
<td>propertiesFrom(TabularData table)</td>
</tr>
<tr>
<td>protected static CompositeData</td>
<td>propertyData(String key, String value, String type)</td>
</tr>
<tr>
<td>static TabularData</td>
<td>tableFrom(java.util.Dictionary properties)</td>
</tr>
<tr>
<td>static TabularData</td>
<td>tableFrom(ServiceReference ref)</td>
</tr>
<tr>
<td>protected static String</td>
<td>typeOf(Class&lt;?&gt; clazz)</td>
</tr>
</tbody>
</table>

### Methods inherited from class Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Field Detail

#### KEY

public static final String **KEY**

See Also:
Constant Field Values
public static final String VALUE
See Also:
Constant Field Values

public static final String TYPE
See Also:
Constant Field Values

public static final String[] PROPERTIES

public static final CompositeType PROPERTY

public static final TabularType PROPERTY_TABLE

protected static final java.util.Set<String> SCALAR_TYPES

protected static final java.util.Set<String> PRIMITIVE_TYPES

Constructor Detail
OSGiProperties

public OSGiProperties()

Method Detail

public static TabularData tableFrom(java.util.Dictionary properties)
**tableFrom**

public static TabularData tableFrom(ServiceReference ref)

**encode**

public static CompositeData encode(String key, Object value)

**propertiesFrom**

public static java.util.Hashtable<String, Object> propertiesFrom(TabularData table)

**encodeArray**

protected static CompositeData encodeArray(String key, Object value, Class<?> componentClazz)

**encodeVector**

protected static CompositeData encodeVector(String key, java.util.Vector value)

**typeOf**

protected static String typeOf(Class<?> clazz)

**propertyData**

protected static CompositeData propertyData(String key, String value,
parse

public static Object parse(String value,
                          String type)

parseArray

protected static Object parseArray(String value,
                           java.util.StringTokenizer tokens)

parseScalarArray

protected static Object parseScalarArray(String value,
                          String type)

createScalarArray

protected static Object[] createScalarArray(String type,
                           int size)

parsePrimitiveArray

protected static Object parsePrimitiveArray(String value,
                           String type)

parseVector

protected static Object parseVector(String value,
                           java.util.StringTokenizer tokens)

parseScalar

protected static Object parseScalar(String value,
                           String type)
5.5.9 OSGiRole

org.osgi.jmx.codec Class OSGiRole

Object
  org.osgi.jmx.codec.OSGiRole

public class OSGiRole
  extends Object

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>protected String</td>
<td></td>
</tr>
<tr>
<td>properties</td>
<td>protected java.util.Hashtable&lt;String, Object&gt;</td>
<td></td>
</tr>
<tr>
<td>ROLE</td>
<td>static CompositeType</td>
<td>OSGiRole</td>
</tr>
<tr>
<td>type</td>
<td>protected int</td>
<td></td>
</tr>
</tbody>
</table>

Constructor Summary

OSGiRole(CompositeData data)

OSGiRole(Role role)

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asCompositeData()</td>
<td>CompositeData</td>
<td></td>
</tr>
<tr>
<td>getName()</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>getProperties()</td>
<td>java.util.Map&lt;String, Object&gt;</td>
<td></td>
</tr>
<tr>
<td>getType()</td>
<td>int</td>
<td></td>
</tr>
</tbody>
</table>

Methods inherited from class Object
cloned, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

name

protected String name
type

protected int type

properties

protected java.util.Hashtable<String, Object> properties

ROLE

public static final CompositeType ROLE

Constructor Detail

OSGiRole

public OSGiRole(CompositeData data)

OSGiRole

public OSGiRole(Role role)

Method Detail

asCompositeData

public CompositeData asCompositeData() throws OpenDataException

Throws:
OpenDataException

getName

public String getName() Returns:
the name

getType

public int getType() Returns:
the type

getProperties

```java
public java.util.Map<String, Object> getProperties()
```

**Returns:**
the credentials

### 5.5.10 OSGiService

**Class OSGiService**

```java
org.osgi.jmx.codec

public class OSGiService
extends Object
```

This class represents the CODEC for the composite data representing an OSGi ServiceReference. It serves as both the documentation of the type structure and as the codification of the mechanism to convert to/from the CompositeData.

The structure of the composite data is:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectClass</td>
<td>Array of String</td>
</tr>
<tr>
<td>BundleIdentifier</td>
<td>long</td>
</tr>
<tr>
<td>UsingBundles</td>
<td>Array of long</td>
</tr>
</tbody>
</table>

---

**Field Summary**

<table>
<thead>
<tr>
<th>static CompositeType</th>
<th>SERVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The CompositeType representation of the service</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static TabularType</th>
<th>SERVICE_TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The TabularType representation of a list of services</td>
<td></td>
</tr>
</tbody>
</table>

---

**Constructor Summary**

```java
public OSGiService(CompositeData data)
Construct an OSGiService encoded in the CompositeData
```

```java
public OSGiService(long identifier, String[] interfaces,
long bundle, long[] usingBundles)
Construct an OSGiService
```
**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompositeData</td>
<td>asCompositeData() Answer the receiver encoded as CompositeData</td>
</tr>
<tr>
<td>long</td>
<td>getBundle()</td>
</tr>
<tr>
<td>long</td>
<td>getIdentifier()</td>
</tr>
<tr>
<td>String[]</td>
<td>getInterfaces()</td>
</tr>
<tr>
<td>long[]</td>
<td>getUsingBundles()</td>
</tr>
<tr>
<td>static TabularData</td>
<td>tableFrom(java.util.ArrayList&lt;OSGiService&gt; services) Construct the TabularData representing a list of services</td>
</tr>
</tbody>
</table>

**Field Detail**

**SERVICE**

public static final CompositeType SERVICE
The CompositeType representation of the service

**SERVICE_TABLE**

public static final TabularType SERVICE_TABLE
The TabularType representation of a list of services

**Constructor Detail**

OSGiService

public OSGiService(CompositeData data)
Construct an OSGiService encoded in the CompositeData
Parameters:
data - - the CompositeData encoding the OSGiService

OSGiService

public OSGiService(long identifier,  
                    String[] interfaces,  
                    long bundle,  
                    long[] usingBundles)

Construct an OSGiService

Parameters:
identifier - 
interfaces - 
bundle - 
usingBundles -

OSGiService

public OSGiService(ServiceReference reference)

Construct an OSGiService from the underlying ServiceReference

Parameters:
reference - - the reference of the service

Method Detail

Table from

public static TabularData  
tableFrom(java.util.ArrayList<OSGiService> services)

Construct the TabularData representing a list of services

Parameters:
services - - the list of services

Returns:
the TabularData representing the list of OSGiServices

AsCompositeData

public CompositeData asCompositeData()

Answer the receiver encoded as CompositeData

Returns:
the CompositeData encoding of the receiver.
getBundle

public long getBundle()

Returns:
the identifier of the bundle the service belongs to

getIdentifier

public long getIdentifier()

Returns:
the identifier of the service

getInterfaces

public String[] getInterfaces()

Returns:
the interfaces implemented by the service

getUsingBundles

public long[] getUsingBundles()

Returns:
the identifiers of the bundles which are using the service

5.5.11 OSGiServiceEvent

org.osgi.jmx.codec Class OSGiServiceEvent

Object
org.osgi.jmx.codec.OSGiServiceEvent

public class OSGiServiceEvent
extends Object

Author:
Hal Hildebrand Date: Nov 24, 2008 Time: 2:42:48 PM
This class represents the CODEC for the composite data representing a OSGi ServiceEvent

It serves as both the documentation of the type structure and as the codification of the mechanism to convert to/from the CompositeData.

The structure of the composite data is:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>BundleIdentifier</td>
<td>long</td>
</tr>
<tr>
<td>BundleLocation</td>
<td>String</td>
</tr>
<tr>
<td>ObjectClass</td>
<td>Array of String</td>
</tr>
</tbody>
</table>
Field Summary

<table>
<thead>
<tr>
<th>static CompositeType</th>
<th>SERVICE_EVENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>The CompositeType representation of the OSGiServiceEvent</td>
<td></td>
</tr>
</tbody>
</table>

Constructor Summary

| OSGiServiceEvent(CompositeData data) |
| Construct an OSGiServiceEvent from the CompositeData representing the event |

| OSGiServiceEvent(long serviceId, long bundleId, String location, String[] interfaces, int eventType) |
| Construct and OSGiServiceEvent |

| OSGiServiceEvent(ServiceEvent event) |

Method Summary

<table>
<thead>
<tr>
<th>CompositeData</th>
<th>asCompositeData()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer the receiver encoded as CompositeData</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>long</th>
<th>getBundleId()</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>getEventType()</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>String[]</th>
<th>getInterfaces()</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>getLocation()</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>long</th>
<th>getServiceId()</th>
</tr>
</thead>
</table>

Methods inherited from class Object

clon, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

SERVICE_EVENT

class public static final CompositeType SERVICE_EVENT
The CompositeType representation of the OSGiServiceEvent
Constructor Detail

OSGiServiceEvent

public OSGiServiceEvent(CompositeData data)
Construct an OSGiServiceEvent from the CompositeData representing the event

Parameters:
data - the CompositeData representation of the event

OSGiServiceEvent

public OSGiServiceEvent(long serviceId,
                          long bundleId,
                          String location,
                          String[] interfaces,
                          int eventType)
Construct and OSGiServiceEvent

Parameters:
serviceId -
bundleId -
location -
interfaces -
eventType -

OSGiServiceEvent

public OSGiServiceEvent(ServiceEvent event)

Method Detail

asCompositeData

public CompositeData asCompositeData()
Answer the receiver encoded as CompositeData

Returns:
the CompositeData encoding of the receiver.

getBundleId

public long getBundleId()
Returns:
the identifier of the bundle the service belongs to
**getEventType**

```java
public int getEventType()
```

**Returns:**
the type of the event

**getInterfaces**

```java
public String[] getInterfaces()
```

**Returns:**
the interfaces the service implements

**getLocation**

```java
public String getLocation()
```

**Returns:**
the location of the bundle the service belongs to

**getServiceImpl**

```java
public long getServiceId()
```

**Returns:**
the identifier of the service

### 5.5.12 OSGiUser

**org.osgi.jmx.codec**

**Class OSGiUser**

**Object**

```java
org.osgi.jmx.codec.OSGiUser
```

**public class OSGiUser**

**extends** Object

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>protected</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.Hashtable&lt;String,Object&gt; credentials</td>
<td></td>
</tr>
<tr>
<td>protected OSGiRole role</td>
<td></td>
</tr>
</tbody>
</table>

**Constructor Summary**

**OSGiUser**(CompositeData data)
**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompositeData</td>
<td>asCompositeData()</td>
</tr>
<tr>
<td>java.util.Map&lt;String, Object&gt;</td>
<td>getCredentials()</td>
</tr>
<tr>
<td>OSGiRole</td>
<td>getRole()</td>
</tr>
</tbody>
</table>

**Methods inherited from class Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Field Detail**

**role**

protected OSGiRole role

**credentials**

protected java.util.Hashtable<String, Object> credentials

**USER**

public static final CompositeType USER

**Constructor Detail**

**OSGiUser**

public OSGiUser(User user)

**OSGiUser**

public OSGiUser(CompositeData data)

**Method Detail**

asCompositeData
public CompositeData asCompositeData() throws OpenDataException

getRole

public OSGiRole getRole()

Returns:
the role

getCredentials

public java.util.Map<String,Object> getCredentials()

Returns:
the credentials

5.5.13 Util

org.osgi.jmx.codec Class Util

Object
org.osgi.jmx.codec.Util

public class Util
extends Object

Author:
Hal Hildebrand
Date: Nov 24, 2008
Time: 7:09:25 AM
Static utilities used by the system

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>static ArrayType</td>
</tr>
<tr>
<td>static ArrayType</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Util()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>static long[] bundleIds(Bundle[] bundles)</td>
</tr>
<tr>
<td>Method</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>static long[]</td>
</tr>
<tr>
<td>static long[]</td>
</tr>
<tr>
<td>static String[]</td>
</tr>
<tr>
<td>static long[]</td>
</tr>
<tr>
<td>static java.util.Map&lt;String, String&gt;</td>
</tr>
<tr>
<td>static String[]</td>
</tr>
<tr>
<td>static long[]</td>
</tr>
<tr>
<td>static String</td>
</tr>
<tr>
<td>static RequiredBundle</td>
</tr>
<tr>
<td>static boolean</td>
</tr>
<tr>
<td>static boolean</td>
</tr>
<tr>
<td>static boolean</td>
</tr>
<tr>
<td>static boolean</td>
</tr>
<tr>
<td>static long[]</td>
</tr>
<tr>
<td>static Long[]</td>
</tr>
<tr>
<td>static String</td>
</tr>
</tbody>
</table>
static long[] serviceIds(ServiceReference[] refs)

Methods inherited from class Object
clone, equals, finalize, getClass, hashCode, notify,
notifyAll, toString, wait, wait, wait

Field Detail
LONG_ARRAY_TYPE

public static ArrayType LONG_ARRAY_TYPE

STRING_ARRAY_TYPE

public static ArrayType STRING_ARRAY_TYPE

Constructor Detail
Util

public Util()

Method Detail
bundleIds

public static long[] bundleIds(Bundle[] bundles)

bundleIds

public static long[] bundleIds(RequiredBundle[] bundles)

getBundlesRequiring

public static long[] getBundlesRequiring(Bundle b,
BundleContext bc,
PackageAdmin admin)

getBundleExportedPackages
public static String[] getBundleExportedPackages(Bundle b, PackageAdmin admin)

getBundleFragments

public static long[] getBundleFragments(Bundle b, PackageAdmin admin)

getBundleHeaders

public static java.util.Map<String,String> getBundleHeaders(Bundle b)

getBundleImportedPackages

public static String[] getBundleImportedPackages(Bundle b, BundleContext bc, PackageAdmin admin)

getBundleDependencies

public static long[] getBundleDependencies(Bundle bundle, PackageAdmin admin)

getBundleState

public static String getBundleState(Bundle b)

getRequiredBundle

public static RequiredBundle getRequiredBundle(Bundle bundle, BundleContext bc, PackageAdmin admin)

isBundleFragment
public static boolean isBundleFragment(Bundle bundle, PackageAdmin admin)

isBundlePersistentlyStarted

public static boolean isBundlePersistentlyStarted(Bundle bundle, StartLevel sl)

isBundleRequired

public static boolean isBundleRequired(Bundle bundle, BundleContext bc, PackageAdmin admin)

isRequiredBundleRemovalPending

public static boolean isRequiredBundleRemovalPending(Bundle bundle, BundleContext bc, PackageAdmin admin)

packageString

public static String packageString(ExportedPackage pkg)

serviceIds

public static long[] serviceIds(ServiceReference[] refs)

LongArrayFrom

public static Long[] LongArrayFrom(long[] array)

longArrayFrom

public static long[] longArrayFrom(Long[] array)
5.6 Core Command and Control Interfaces

These interfaces provide the management client with batch operations on the core framework APIs as well as access to the core OSGi artifacts, Bundles, Services and Packages.

5.6.1 Interface FrameworkMBean

org.osgi.jmx.core Interface FrameworkMBean

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.

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static String[]</td>
<td>BUNDLE_ACTION_RESULT</td>
</tr>
<tr>
<td>static String</td>
<td>BUNDLE_BATCH_ACTION_RESULT</td>
</tr>
<tr>
<td>static String</td>
<td>BUNDLE_BATCH_INSTALL_RESULT</td>
</tr>
<tr>
<td>static String</td>
<td>BUNDLE.Completed</td>
</tr>
<tr>
<td>static String</td>
<td>BUNDLE_ERROR_MESSAGE</td>
</tr>
<tr>
<td>static String</td>
<td>BUNDLE_IN_ERROR</td>
</tr>
<tr>
<td>static String</td>
<td>BUNDLE_REMAINING</td>
</tr>
<tr>
<td>static String</td>
<td>BUNDLE_SUCCESS</td>
</tr>
</tbody>
</table>
### Method Summary

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><code>getFrameworkStartLevel()</code></td>
<td>Retrieve the framework start level</td>
</tr>
<tr>
<td>int</td>
<td><code>getInitialBundleStartLevel()</code></td>
<td>Answer the initial start level assigned to a bundle when it is first started</td>
</tr>
<tr>
<td>long</td>
<td><code>installBundle(String location)</code></td>
<td>Install the bundle indicated by the bundleLocations</td>
</tr>
<tr>
<td>long</td>
<td><code>installBundle(String location, String url)</code></td>
<td>Install the bundle indicated by the bundleLocations</td>
</tr>
<tr>
<td>CompositeData</td>
<td><code>installBundles(String[] locations)</code></td>
<td>Batch install the bundles indicated by the list of bundleLocationUrls</td>
</tr>
<tr>
<td>CompositeData</td>
<td><code>installBundles(String[] locations, String[] urls)</code></td>
<td>Batch install the bundles indicated by the list of bundleLocationUrls</td>
</tr>
<tr>
<td>void</td>
<td><code>refreshPackages(long bundleIdentifier)</code></td>
<td>Force the update, replacement or removal of the packages identified by the list of bundles</td>
</tr>
<tr>
<td>CompositeData</td>
<td><code>refreshPackages(long[] bundleIdentifiers)</code></td>
<td>Force the update, replacement or removal of the packages identified by the list of bundles</td>
</tr>
<tr>
<td>boolean</td>
<td><code>resolveBundle(long bundleIdentifier)</code></td>
<td>Resolve the bundle indicated by the unique symbolic name and version</td>
</tr>
<tr>
<td>boolean</td>
<td><code>resolveBundles(long[] bundleIdentifiers)</code></td>
<td>Batch resolve the bundles indicated by the list of bundle identifiers</td>
</tr>
<tr>
<td>void</td>
<td><code>restartFramework()</code></td>
<td>Restart the framework by updating the system bundle</td>
</tr>
<tr>
<td>void</td>
<td><code>setBundleStartLevel(long bundleIdentifier, int newlevel)</code></td>
<td>Set the start level for the bundle</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><code>CompositeData.setBundleStartLevels()</code></td>
<td>Set the start levels for the list of bundles</td>
<td></td>
</tr>
<tr>
<td><code>void setFrameworkStartLevel()</code></td>
<td>Set the start level for the framework</td>
<td></td>
</tr>
<tr>
<td><code>void setInitialBundleStartLevel()</code></td>
<td>Set the initial start level assigned to a bundle when it is first started</td>
<td></td>
</tr>
<tr>
<td><code>void shutdownFramework()</code></td>
<td>Shutdown the framework by stopping the system bundle</td>
<td></td>
</tr>
<tr>
<td><code>void startBundle()</code></td>
<td>Start the bundle indicated by the bundle identifier</td>
<td></td>
</tr>
<tr>
<td><code>CompositeData.startBundles()</code></td>
<td>Batch start the bundles indicated by the list of bundle identifier</td>
<td></td>
</tr>
<tr>
<td><code>void stopBundle()</code></td>
<td>Stop the bundle indicated by the bundle identifier</td>
<td></td>
</tr>
<tr>
<td><code>CompositeData.stopBundles()</code></td>
<td>Batch stop the bundles indicated by the list of bundle identifier</td>
<td></td>
</tr>
<tr>
<td><code>void uninstallBundle()</code></td>
<td>Uninstall the bundle indicated by the bundle identifier</td>
<td></td>
</tr>
<tr>
<td><code>CompositeData.uninstallBundles()</code></td>
<td>Batch uninstall the bundles indicated by the list of bundle identifiers</td>
<td></td>
</tr>
<tr>
<td><code>void updateBundle()</code></td>
<td>Update the bundle indicated by the bundle identifier</td>
<td></td>
</tr>
<tr>
<td><code>void updateBundle()</code></td>
<td>Update the bundle identified by the bundle identifier</td>
<td></td>
</tr>
<tr>
<td><code>CompositeData.updateBundles()</code></td>
<td>Batch update the bundles indicated</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © Oracle Corporation 2009 All Rights Reserved
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateBundles(long[] bundleIdentifiers, String[] urls)</td>
<td>Update the bundle uniquely identified by the bundle symbolic name and version using the contents of the supplied urls</td>
</tr>
<tr>
<td>updateFramework()</td>
<td>Update the framework by updating the system bundle</td>
</tr>
</tbody>
</table>

### Field Detail

#### BUNDLE_COMPLETED

static final String BUNDLE_COMPLETED

The name of the item containing the list of bundles completing the batch operation in the CompositeData

See Also:
Constant Field Values

#### BUNDLE_ERROR_MESSAGE

static final String BUNDLE_ERROR_MESSAGE

The name of the item containing the error message of the batch operation in the CompositeData

See Also:
Constant Field Values

#### BUNDLE_IN_ERROR

static final String BUNDLE_IN_ERROR

The name of the item containing the bundle which caused the error during the batch operation in the CompositeData

See Also:
Constant Field Values

#### BUNDLE_REMAINING

static final String BUNDLE_REMAINING

The name of the item containing the list of remaining bundles unprocessed by the failing batch operation in the CompositeData

Copyright © Oracle Corporation 2009 All Rights Reserved
See Also:
Constant Field Values

**BUNDLE_SUCCESS**

static final String BUNDLE_SUCCESS
The name of the item containing the success status of the batch operation in the CompositeData

See Also:
Constant Field Values

**BUNDLE_ACTION_RESULT**

static final String[] BUNDLE_ACTION_RESULT
The item names in the CompositeData representing the result of a batch operation

**BUNDLE_BATCH_ACTION_RESULT**

static final String BUNDLE_BATCH_ACTION_RESULT
The name of the CompositeType which represents the result of a batch operation

See Also:
Constant Field Values

**BUNDLE_BATCH_INSTALL_RESULT**

static final String BUNDLE_BATCH_INSTALL_RESULT
The name of the CompositeType which represents the result of a batch install operation

See Also:
Constant Field Values

### Method Detail

#### getFrameworkStartLevel

```java
int getFrameworkStartLevel() throws IOException
```

Retrieve the framework start level

**Returns:**
the framework start level

Throws:
IOException - if the operation failed

getInitialBundleStartLevel

```java
int getInitialBundleStartLevel() throws IOException
```

Answer the initial start level assigned to a bundle when it is first started

Returns:
the start level

Throws:
IOException - if the operation failed

installBundle

```java
long installBundle(String location) throws IOException
```

Install the bundle indicated by the bundle Locations

Parameters:
location - - the location of the bundle to install

Returns:
the bundle id the installed bundle

Throws:
IOException - if the operation does not succeed

installBundle

```java
long installBundle(String location, String url) throws IOException
```

Install the bundle indicated by the bundle Locations

Parameters:
location - - the location to assign to the bundle
url - - the URL which will supply the bytes for the bundle

Returns:
the bundle id the installed bundle

Throws:
IOException - if the operation does not succeed

installBundles

```java
CompositeData installBundles(String[] locations)
```
installBundles

CompositeData installBundles(String[] locations, String[] urls) throws IOException

Batch install the bundles indicated by the list of bundleLocationUrls

Parameters:
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:
BatchBundleResult for the precise specification of the CompositeData type representing the returned result.

refreshPackages

void refreshPackages(long bundleIdentifier) throws IOException

Force the update, replacement or removal of the pacakges identified by the list of bundles
Parameters:
bundleIdentifier -- the bundle identifier

Throws:
IOException - if the operation failed

See Also:
BundleBatchActionResult for the precise specification of the
CompositeData type representing the returned result.

refreshPackages

CompositeData refreshPackages(long[] bundleIdentifiers) throws IOException

Force the update, replacement or removal of the packages identified by the list of bundles

Parameters:
bundleIdentifiers -- the array of bundle identifiers

Returns:
the resulting state from executing the operation

Throws:
IOException - if the operation failed

See Also:
BundleBatchActionResult for the precise specification of the
CompositeData type representing the returned result.

resolveBundle

boolean resolveBundle(long bundleIdentifier) throws IOException

Resolve the bundle indicated by the unique symbolic name and version

Parameters:
bundleIdentifier -- the bundle identifier

Returns:
true if the bundle was resolved, false otherwise

Throws:
IOException - if the operation does not succeed
IllegalArgumentException - if the bundle indicated does not exist

resolveBundles
boolean `resolveBundles(long[] bundleIdentifiers)`

Batch resolve the bundles indicated by the list of bundle identifiers

**Parameters:**
bundleIdentifiers - the identifiers of the bundles to resolve

**Returns:**
true if the bundles were resolved, false otherwise

**Throws:**
IOException - if the operation does not succeed

`restartFramework`

`void restartFramework() throws IOException`

Restart the framework by updating the system bundle

**Throws:**
IOException - if the operation failed

`setBundleStartLevel`

`void setBundleStartLevel(long bundleIdentifier, int newlevel) throws IOException`

Set the start level for the bundle identifier

**Parameters:**
bundleIdentifier -- the bundle identifier
newlevel -- the new start level for the bundle

**Throws:**
IOException - if the operation failed

`setBundleStartLevels`

`CompositeData setBundleStartLevels(long[] bundleIdentifiers, int[] newlevels) throws IOException`

Set the start levels for the list of bundles

**Parameters:**
bundleIdentifiers -- the array of bundle identifiers
newlevels -- the array of new start level for the bundles

Returns: 
the resulting state from executing the operation

Throws: 
IOException - if the operation failed

See Also: 
BundleBatchActionResult for the precise specification of the
CompositeData type representing the returned result.

```java
setFrameworkStartLevel

void setFrameworkStartLevel(int newlevel)
    throws IOException

Set the start level for the framework

Parameters:
newlevel -- the new start level

Throws:
IOException - if the operation failed

setInitialBundleStartLevel

void setInitialBundleStartLevel(int newlevel)
    throws IOException

Set the initial start level assigned to a bundle when it is first started

Parameters:
newlevel -- the new start level

Throws:
IOException - if the operation failed

shutdownFramework

void shutdownFramework()
    throws IOException

Shutdown the framework by stopping the system bundle

Throws:
IOException - if the operation failed

startBundle

void startBundle(long bundleIdentifier)
```
throws IOException
Start the bundle indicated by the bundle identifier

Parameters:
bundleIdentifier -- the bundle identifier

Throws:
IOException - if the operation does not succeed
IllegalArgumentException - if the bundle indicated does not exist

startBundles

CompositeData startBundles(long[] bundleIdentifiers)

throws IOException
Batch start the bundles indicated by the list of bundle identifier

Parameters:
bundleIdentifiers -- the array of bundle identifiers

Returns:
the resulting state from executing the operation

Throws:
IOException - if the operation does not succeed

See Also:
BundleBatchActionResult for the precise specification of the CompositeData type representing the returned result.

stopBundle

void stopBundle(long bundleIdentifier)

throws IOException
Stop the bundle indicated by the bundle identifier

Parameters:
bundleIdentifier -- the bundle identifier

Throws:
IOException - if the operation does not succeed
IllegalArgumentException - if the bundle indicated does not exist

stopBundles

CompositeData stopBundles(long[] bundleIdentifiers)
Batch stop the bundles indicated by the list of bundle identifiers

**Parameters:**
- bundleIdentifiers -- the array of bundle identifiers

**Returns:**
- the resulting state from executing the operation

**Throws:**
- IOException - if the operation does not succeed

**See Also:**
- BundleBatchActionResult for the precise specification of the CompositeData type representing the returned result.

---

### uninstallBundle

```java
void uninstallBundle(long bundleIdentifier) throws IOException
```

Uninstall the bundle indicated by the bundle identifier

**Parameters:**
- bundleIdentifier -- the bundle identifier

**Throws:**
- IOException - if the operation does not succeed
- IllegalArgumentException - if the bundle indicated does not exist

### uninstallBundles

```java
CompositeData uninstallBundles(long[] bundleIdentifiers) throws IOException
```

Batch uninstall the bundles indicated by the list of bundle identifiers

**Parameters:**
- bundleIdentifiers -- the array of bundle identifiers

**Returns:**
- the resulting state from executing the operation

**Throws:**
- IOException - if the operation does not succeed

**See Also:**
BundleBatchActionResult for the precise specification of the CompositeData type representing the returned result.

updateBundle

void updateBundle(long bundleIdentifier)
    throws IOException
Update the bundle indicated by the bundle identifier

Parameters:
bundleIdentifier -- the bundle identifier

Throws:
IOException - if the operation does not succeed
IllegalArgumentException - if the bundle indicated does not exist

updateBundle

void updateBundle(long bundleIdentifier, String url)
    throws IOException
Update the bundle identified by the bundle identifier

Parameters:
bundleIdentifier -- the bundle identifier
url -- the URL to use to update the bundle

Throws:
IOException - if the operation does not succeed
IllegalArgumentException - if the bundle indicated does not exist

updateBundles

CompositeData updateBundles(long[] bundleIdentifiers)
    throws IOException
Batch update the bundles indicated by the list of bundle identifier

Parameters:
bundleIdentifiers -- the array of bundle identifiers

Returns:
the resulting state from executing the operation

Throws:
IOException - if the operation does not succeed
See Also: 
`BundleBatchActionResult for the precise specification of the CompositeData type representing the returned result.`

**updateBundles**

```java
CompositeData updateBundles(long[] bundleIdentifiers, String[] urls) throws IOException
```

Update the bundle uniquely identified by the bundle symbolic name and version using the contents of the supplied urls

**Parameters:**
- `bundleIdentifiers` -- the array of bundle identifiers
- `urls` -- the array of URLs to use to update the bundles

**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:
`BundleBatchActionResult for the precise specification of the CompositeData type representing the returned result.`

**updateFramework**

```java
void updateFramework() throws IOException
```

Update the framework by updating the system bundle

**Throws:**
- `IOException` - if the operation failed
5.6.2 Interface BundleStateMBean

org.osgi.jmx.core Interface BundleStateMBean

```
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.

See OSGiBundleEvent for the precise definition of the CompositeData for the notification sent.

### Field Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static String[]</code></td>
<td><code>BUNDLE</code></td>
</tr>
<tr>
<td><code>static String[]</code></td>
<td><code>BUNDLE_EVENT</code></td>
</tr>
<tr>
<td><code>static String</code></td>
<td><code>BUNDLE_EVENT_TYPE</code></td>
</tr>
<tr>
<td><code>static String</code></td>
<td><code>BUNDLE_EXPORTED_PACKAGES</code></td>
</tr>
<tr>
<td><code>static String</code></td>
<td><code>BUNDLE_FRAGMENT</code></td>
</tr>
<tr>
<td><code>static String</code></td>
<td><code>BUNDLE_FRAGMENTS</code></td>
</tr>
<tr>
<td><code>static String</code></td>
<td><code>BUNDLE_HEADER_TYPE</code></td>
</tr>
<tr>
<td><code>static String</code></td>
<td><code>BUNDLE_HEADERS</code></td>
</tr>
<tr>
<td><code>static String</code></td>
<td><code>BUNDLE_HEADERS_TYPE</code></td>
</tr>
<tr>
<td><code>static String</code></td>
<td><code>BUNDLE_HOSTS</code></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE ID</strong></td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_IMPORTED_PACKAGES</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_LAST_MODIFIED</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_LOCATION</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_PERSISTENTLY_STARTED</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_REGISTERED_SERVICES</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_REMOVAL_PENDING</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_REQUIRED</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_REQUIRED_BUNDLES</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_REQUIRING_BUNDLES</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_SERVICES_IN_USE</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_START_LEVEL</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_STATE</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_SYMBOLIC_NAME</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>BUNDLE_TYPE_NAME</strong></td>
</tr>
<tr>
<td>static String</td>
<td><strong>EVENT_TYPE</strong></td>
</tr>
<tr>
<td>Method Summary</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>TabularData</td>
<td><strong>getBundles()</strong></td>
</tr>
<tr>
<td>long[]</td>
<td><strong>getDependencies(long bundleIdentifier)</strong></td>
</tr>
<tr>
<td>String[]</td>
<td><strong>getExportedPackages(long bundleId)</strong></td>
</tr>
<tr>
<td>long[]</td>
<td><strong>getFragments(long bundleId)</strong></td>
</tr>
<tr>
<td>TabularData</td>
<td><strong>getHeaders(long bundleId)</strong></td>
</tr>
<tr>
<td>long[]</td>
<td><strong>getHosts(long fragment)</strong></td>
</tr>
<tr>
<td>String[]</td>
<td><strong>getImportedPackages(long bundleId)</strong></td>
</tr>
<tr>
<td>long</td>
<td><strong>getLastModified(long bundleId)</strong></td>
</tr>
<tr>
<td>long[]</td>
<td><strong>getRegisteredServices(long bundleId)</strong></td>
</tr>
<tr>
<td>long[]</td>
<td><strong>getRequiringBundles(long bundleIdentifier)</strong></td>
</tr>
<tr>
<td>long[]</td>
<td><strong>getServicesInUse(long bundleIdentifier)</strong></td>
</tr>
<tr>
<td>int</td>
<td><strong>getStartLevel(long bundleId)</strong></td>
</tr>
<tr>
<td>String</td>
<td><strong>getState(long bundleId)</strong></td>
</tr>
</tbody>
</table>
Field Detail

**BUNDLE_EVENT_TYPE**

```java
static final String BUNDLE_EVENT_TYPE
    The type of the event which is emitted when bundle state changes occur in the OSGi container
```

See Also:
- [Constant Field Values](#)

**BUNDLE_EXPORTED_PACKAGES**

```java
static final String BUNDLE_EXPORTED_PACKAGES
    The name of the item containing the exported packages in the CompositeData
```

See Also:
- [Constant Field Values](#)

**BUNDLE_FRAGMENT**

```java
static final String BUNDLE_FRAGMENT
    The name of the item containing the fragment status in the CompositeData
```

See Also:
Constant Field Values

BUNDLE_FRAGMENTS

static final String BUNDLE_FRAGMENTS
The name of the item containing the list of fragments the bundle is host to in the CompositeData representing a Bundle

See Also:
Constant Field Values

BUNDLE_HEADERS

static final String BUNDLE_HEADERS
The name of the item containing the bundle headers in the CompositeData

See Also:
Constant Field Values

BUNDLE_HOSTS

static final String BUNDLE_HOSTS
The name of the item containing the bundle identifiers representing the hosts

See Also:
Constant Field Values

BUNDLE_ID

static final String BUNDLE_ID
The name of the item containing the bundle identifier in the CompositeData

See Also:
Constant Field Values

BUNDLE_IMPORTED_PACKAGES

static final String BUNDLE_IMPORTED_PACKAGES
The name of the item containing the imported packages in the CompositeData

See Also:
Constant Field Values
BUNDLE_LAST_MODIFIED

static final String BUNDLE_LAST_MODIFIED
The name of the item containing the last modified time in the CompositeData

See Also:
Constant Field Values

BUNDLE_LOCATION

static final String BUNDLE_LOCATION
The name of the item containing the bundle location in the CompositeData

See Also:
Constant Field Values

BUNDLE_PERSISTENTLY_STARTED

static final String BUNDLE_PERSISTENTLY_STARTED
The name of the item containing the indication of persistently started in the CompositeData

See Also:
Constant Field Values

BUNDLE_REGISTERED_SERVICES

static final String BUNDLE_REGISTERED_SERVICES
The name of the item containing the registered services of the bundle in the CompositeData

See Also:
Constant Field Values

BUNDLE_REMOVAL_PENDING

static final String BUNDLE_REMOVAL_PENDING
The name of the item containing the indication of removal pending in the CompositeData

See Also:
Constant Field Values

BUNDLE_REQUIRED

static final String BUNDLE_REQUIRED
The name of the item containing the required status in the CompositeData

See Also:
Constant Field Values

**BUNDLE_REQUIRED_BUNDLES**

static final String **BUNDLE_REQUIRED_BUNDLES**
The name of the item containing the required bundles in the CompositeData

See Also:
Constant Field Values

**BUNDLE_REQUIRING_BUNDLES**

static final String **BUNDLE_REQUIRING_BUNDLES**
The name of the item containing the bundles requiring this bundle in the CompositeData

See Also:
Constant Field Values

**BUNDLE_SERVICES_IN_USE**

static final String **BUNDLE_SERVICES_IN_USE**
The name of the item containing the services in use by this bundle in the CompositeData

See Also:
Constant Field Values

**BUNDLE_START_LEVEL**

static final String **BUNDLE_START_LEVEL**
The name of the item containing the start level in the CompositeData

See Also:
Constant Field Values

**BUNDLE_STATE**

static final String **BUNDLE_STATE**
The name of the item containing the bundle state in the CompositeData
See Also:
Constant Field Values

**BUNDLE_SYMBOLIC_NAME**

```
static final String BUNDLE_SYMBOLIC_NAME
The name of the item containing the symbolic name in the CompositeData
```

See Also:
Constant Field Values

**BUNDLE_TYPE_NAME**

```
static final String BUNDLE_TYPE_NAME
The name CompositeData type for a bundle
```

See Also:
Constant Field Values

**EVENT_TYPE**

```
static final String EVENT_TYPE
The name of the item containing the event type in the CompositeData
```

See Also:
Constant Field Values

**BUNDLE**

```
static final String[] BUNDLE
The item names in the CompositeData representing an OSGi Bundle
```

**BUNDLE_EVENT**

```
static final String[] BUNDLE_EVENT
The item names in the CompositeData representing the event raised for bundle events within the OSGi container by this bean
```
**BUNDLE_HEADERS_TYPE**

static final String BUNDLE_HEADERS_TYPE

See Also:
Constant Field Values

**BUNDLE_HEADER_TYPE**

static final String BUNDLE_HEADER_TYPE

See Also:
Constant Field Values

## Method Detail

### getDependencies

```java
long[] getDependencies(long bundleIdentifier) throws IOException
```

Answer the list of identifiers of the bundles this bundle depends upon

**Parameters:**
- bundleIdentifier -- the bundle identifier

**Returns:**
- the list of bundle identifiers

**Throws:**
- IOException - if the operation fails
- IllegalArgumentException - if the bundle indicated does not exist

### getBundles

```java
TabularData getBundles() throws IOException
```

Answer the bundle state of the system in tabular form Each row of the returned table represents a single bundle. For each bundle, the following row is returned

- location - String
- bundle identifier - String
- symbolic name - String
- start level - int
- state - String
- last modified - long
- persistently started - boolean
- removal pending - boolean
- required - boolean
- fragment - boolean
- registered services - long[]
- services in use - long[]
- headers - TabularData
- exported packages - String[]
- imported packages - String[]
- fragments - long[]
hosts - long[]
required bundles - long[]
requiring bundles - long[]

**Returns:**
the tabular representation of the bundle state

**Throws:**
IOException

**See Also:**
for the precise specification of the CompositeType definition for each row of the table.

### getExportedPackages

```java
String[] getExportedPackages(long bundleId) throws IOException
```

Answer the list of exported packages for this bundle

**Parameters:**
bundleId -

**Returns:**
the array of package names, combined with their version in the format

**Throws:**
IOException - if the operation fails
IllegalArgumentException - if the bundle indicated does not exist

### getFragments

```java
long[] getFragments(long bundleId) throws IOException
```

Answer the list of the bundle ids of the fragments associated with this bundle

**Parameters:**
bundleId -

**Returns:**
the array of bundle identifiers

**Throws:**
IOException - if the operation fails
IllegalArgumentException - if the bundle indicated does not exist

### getHeaders

```java
TabularData getHeaders(long bundleId) throws IOException
```

Answer the headers for the bundle uniquely identified by the bundle id
**getHosts**

```java
long[] getHosts(long fragment) throws IOException
```

Answer the list of bundle ids of the bundles which host a fragment

**Parameters:**
- `fragment` - the bundle id of the fragment

**Returns:**
- the array of bundle identifiers

**Throws:**
- `IOException` - if the operation fails
- `IllegalArgumentException` - if the bundle indicated does not exist

**getImportedPackages**

```java
String[] getImportedPackages(long bundleId) throws IOException
```

Answer the array of the packages imported by this bundle

**Parameters:**
- `bundleId` - the bundle identifier

**Returns:**
- the array of package names, combined with their version in the format

**Throws:**
- `IOException` - if the operation fails
- `IllegalArgumentException` - if the bundle indicated does not exist

**getLastModified**

```java
long getLastModified(long bundleId) throws IOException
```

Answer the last modified time of a bundle

**Parameters:**
- `bundleId` - the unique identifier of a bundle
Returns:  
the last modified time  

Throws:  
IOException - if the operation fails  
IllegalArgumentException - if the bundle indicated does not exist

getRegisteredServices

long[] getRegisteredServices(long bundleId)
throws IOException

Answer the list of service identifiers representing the services this bundle exports

Parameters:  
bundleId - the bundle identifier  

Returns:  
the list of service identifiers  

Throws:  
IOException - if the operation fails  
IllegalArgumentException - if the bundle indicated does not exist

getRequiringBundles

long[] getRequiringBundles(long bundleIdentifier)
throws IOException

Answer the list of identifiers of the bundles which require this bundle

Parameters:  
bundleIdentifier - the bundle identifier  

Returns:  
the list of bundle identifiers  

Throws:  
IOException - if the operation fails  
IllegalArgumentException - if the bundle indicated does not exist

getServicesInUse

long[] getServicesInUse(long bundleIdentifier)
throws IOException

Answer the list of service identifiers which refer to the the services this bundle is using

Parameters:  
bundleIdentifier - the bundle identifier  

Returns:  
the list of service identifiers  

Throws:  
IOException - if the operation fails
IllegalArgumentException - if the bundle indicated does not exist

**getStartLevel**

```java
int getStartLevel(long bundleId)
throws IOException
```

Answer the start level of the bundle

**Parameters:**
- bundleId - the identifier of the bundle

**Returns:**
- the start level

**Throws:**
- IOException - if the operation fails
- IllegalArgumentException - if the bundle indicated does not exist

**getState**

```java
String getState(long bundleId)
throws IOException
```

Answer the symbolic name of the state of the bundle

**Parameters:**
- bundleId - the identifier of the bundle

**Returns:**
- the string name of the bundle state

**Throws:**
- IOException - if the operation fails
- IllegalArgumentException - if the bundle indicated does not exist

**getSymbolicName**

```java
String getSymbolicName(long bundleId)
throws IOException
```

Answer the symbolic name of the bundle

**Parameters:**
- bundleId - the identifier of the bundle

**Returns:**
- the symbolic name

**Throws:**
- IOException - if the operation fails
- IllegalArgumentException - if the bundle indicated does not exist

**isPersistentlyStarted**
boolean isPersistentlyStarted(long bundleId)
    throws IOException
Answer if the bundle is persistently started when its start level is reached

Parameters:
bundleId - the identifier of the bundle

Returns:
true if the bundle is persistently started

Throws:
IOException - if the operation fails
IllegalArgumentException - if the bundle indicated does not exist

isFragment

boolean isFragment(long bundleId)
    throws IOException
Answer whether the bundle is a fragment or not

Parameters:
bundleId - the identifier of the bundle

Returns:
true if the bundle is a fragment

Throws:
IOException - if the operation fails
IllegalArgumentException - if the bundle indicated does not exist

isRemovalPending

boolean isRemovalPending(long bundleId)
    throws IOException
Answer true if the bundle is pending removal

Parameters:
bundleId - the identifier of the bundle

Returns:
true if the bundle is pending removal

Throws:
IOException - if the operation fails
IllegalArgumentException - if the bundle indicated does not exist

isRequired

boolean isRequired(long bundleId)
    throws IOException
Answer true if the bundle is required by another bundle
Parameters:
bundleId - the identifier of the bundle

Returns:
true if the bundle is required by another bundle

Throws:
IOException - if the operation fails
IllegalArgumentException - if the bundle indicated does not exist
5.6.3 Interface PackageStateMBean

org.osgi.jmx.core Interface PackageStateMBean

public interface PackageStateMBean

Author:
Hal Hildebrand Date: Sep 23, 2008 Time: 9:04:23 AM
This MBean represents the Package state of the framework.

Field Summary

| static String | BUNDLE_IDENTIFIER | The name of the item containing the bundle identifier in the CompositeData |
| static String | IMPORTING_BUNDLES | The name of the item containing the importing bundles in the CompositeData |
| static String[] | PACKAGE | The item names in the CompositeData representing the OSGi Package |
| static String | PACKAGE_NAME | The name of the item containing the package name in the CompositeData |
| static String | PACKAGE_PENDING_REMOVAL | The name of the item containing the pending removal status of the package in the CompositeData |
| static String | PACKAGE_VERSION | The name of the item containing the package version in the CompositeData |

Method Summary

| long | getExportingBundle(String packageName, String version) | Answer the identifier of the bundle exporting the package |
| long[] | getImportingBundles(String packageName, String version) | Answer the list of identifiers of the bundles importing the package |
| TabularData | getPackages() | Answer the package state of the system in tabular form |
### Field Detail

#### BUNDLE_IDENTIFIER

static final String BUNDLE_IDENTIFIER
The name of the item containing the bundle identifier in the CompositeData

**See Also:**
Constant Field Values

#### IMPORTING_BUNDLES

static final String IMPORTING_BUNDLES
The name of the item containing the importing bundles in the CompositeData

**See Also:**
Constant Field Values

#### PACKAGE_NAME

static final String PACKAGE_NAME
The name of the item containing the package name in the CompositeData

**See Also:**
Constant Field Values

#### PACKAGE_PENDING_REMOVAL

static final String PACKAGE_PENDING_REMOVAL
The name of the item containing the pending removal status of the package in the CompositeData

**See Also:**
Constant Field Values

#### PACKAGE_VERSION

static final String PACKAGE_VERSION
The name of the item containing the package version in the CompositeData

See Also:
Constant Field Values

PACKAGE

static final String[] PACKAGE
The item names in the CompositeData representing the OSGi Package

Method Detail

getExportingBundle

long getExportingBundle(String packageName, String version)
throws IOException

Answer the identifier of the bundle exporting the package

Parameters:
packageName -- the package name
version -- the version of the package

Returns:
the bundle identifier or -1 if there is no bundle

Throws:
IOException - if the operation fails
IllegalArgumentException - if the package indicated does not exist

getImportingBundles

long[] getImportingBundles(String packageName, String version)
throws IOException

Answer the list of identifiers of the bundles importing the package

Parameters:
packageName -- the package name
version -- the version of the package

Returns:
the list of bundle identifiers

Throws:
IOException - if the operation fails
IllegalArgumentException - if the package indicated does not exist

getPackages
TabularData **getPackages**()
Answer the package state of the system in tabular form

**Returns:**
the tabular representation of the package state

**Throws:**
IOException

**See Also:**
for the details of the TabularType.

Each row of the returned table represents a single package. For each package, the following row is returned
- name - String
- version - String
- removal pending - boolean
- exporting bundle - long
- importing bundles - long[]

**isRemovalPending**

```java
boolean isRemovalPending(String packageName, String version) throws IOException
```

Answer if this package is exported by a bundle which has been updated or uninstalled

**Parameters:**
packageName - - the package name
version - - the version of the package

**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

### 5.6.4 Interface ServiceStateMBean

```
org.osgi.jmx.core Interface ServiceStateMBean
```

**public interface ServiceStateMBean**

**Author:**
Hal Hildebrand Date: Sep 23, 2008 Time: 8:57:33 AM
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.

See OSGiBundleEvent for the precise definition of the CompositeData for the notification sent.
### Field Summary

<table>
<thead>
<tr>
<th>static String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUNDLE_IDENTIFIER</td>
<td>The name of the item containing the bundle identifier in the CompositeData</td>
</tr>
<tr>
<td>BUNDLE_LOCATION</td>
<td>The name of the item containing the bundle location in the CompositeData</td>
</tr>
<tr>
<td>EVENT_TYPE</td>
<td>The name of the item containing the event type in the CompositeData</td>
</tr>
<tr>
<td>OBJECT_CLASS</td>
<td>The name of the item containing the interfaces of the service in the</td>
</tr>
<tr>
<td></td>
<td>CompositeData</td>
</tr>
<tr>
<td>SERVICE</td>
<td>The item names in the CompositeData representing the service</td>
</tr>
<tr>
<td>SERVICEEvento</td>
<td>The item names in the CompositeData representing the ServiceEvent</td>
</tr>
<tr>
<td>SERVICE_EVENT_TYPE</td>
<td>The type of the JMX event raised in response to ServiceEvent in the</td>
</tr>
<tr>
<td></td>
<td>underlying OSGi container</td>
</tr>
<tr>
<td>SERVICE_ID</td>
<td>The name of the item containing the service identifier in the CompositeData</td>
</tr>
<tr>
<td>USING_BUNDLES</td>
<td>The name of the item containing the bundles using the service in the</td>
</tr>
<tr>
<td></td>
<td>CompositeData</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>getBundle(long serviceId)</td>
<td>Answer the bundle identifier of the bundle which registered the service</td>
</tr>
<tr>
<td>TabularData</td>
<td>getProperties(long serviceId)</td>
<td>Answer the map of credentials associated with this service</td>
</tr>
<tr>
<td>String[]</td>
<td>getServiceInterfaces(long serviceId)</td>
<td>Answer the list of interfaces that this service implements</td>
</tr>
<tr>
<td>TabularData</td>
<td>getServices()</td>
<td></td>
</tr>
<tr>
<td>long[]</td>
<td>getUsingBundles(long service Id)</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>Answer the service state of the system in tabular form</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Answer the list of identifiers of the bundles that use the service</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Field Detail

**BUNDLE_IDENTIFIER**

```java
static final String BUNDLE_IDENTIFIER
```

The name of the item containing the bundle identifier in the CompositeData

See Also:
- [Constant Field Values](#)

**BUNDLE_LOCATION**

```java
static final String BUNDLE_LOCATION
```

The name of the item containing the bundle location in the CompositeData

See Also:
- [Constant Field Values](#)

**EVENT_TYPE**

```java
static final String EVENT_TYPE
```

The name of the item containing the event type in the CompositeData

See Also:
- [Constant Field Values](#)

**OBJECT_CLASS**

```java
static final String OBJECT_CLASS
```

The name of the item containing the interfaces of the service in the CompositeData

See Also:
- [Constant Field Values](#)

**SERVICE_EVENT_TYPE**
static final String \textbf{SERVICE\_EVENT\_TYPE}  
The type of the JMX event raised in response to \texttt{ServiceEvent} in the underlying OSGi container 

\textbf{See Also:}  
\url{Constant Field Values} 

\section*{SERVICE\_ID}  

static final String \textbf{SERVICE\_ID}  
The name of the item containing the service identifier in the \texttt{CompositeData} 

\textbf{See Also:}  
\url{Constant Field Values} 

\section*{USING\_BUNDLES}  

static final String \textbf{USING\_BUNDLES}  
The name of the item containing the bundles using the service in the \texttt{CompositeData} 

\textbf{See Also:}  
\url{Constant Field Values} 

\section*{SERVICE}  

static final String[] \textbf{SERVICE}  
The item names in the \texttt{CompositeData} representing the service 

\section*{SERVICE\_EVENT}  

static final String[] \textbf{SERVICE\_EVENT}  
The item names in the \texttt{CompositeData} representing the \texttt{ServiceEvent} 

\section*{Method Detail}  

\textbf{get\_Service\_Interfaces}  

\begin{verbatim}
String[] getServiceInterfaces(long serviceId) throws IOException
\end{verbatim}  

Answer the list of interfaces that this service implements
Parameters:
serviceId -- the identifier of the service

Returns:
the list of interfaces

Throws:
IOException - if the operation fails
IllegalArgumentException - if the service indicated does not exist

getBundle

long getBundle(long serviceId)
throws IOException
Answer the bundle identifier of the bundle which registered the service

Parameters:
serviceId -- the identifier of the service

Returns:
the identifier for the bundle

Throws:
IOException - if the operation fails
IllegalArgumentException - if the service indicated does not exist

getProperties

TabularData getProperties(long serviceId)
throws IOException
Answer the map of credentials associated with this service

Parameters:
serviceId -- the identifier of the service

Returns:
the table of credentials. These include the standard mandatory service.id and objectClass credentials as defined in the Constants interface

Throws:
IOException - if the operation fails
IllegalArgumentException - if the service indicated does not exist

See Also:
for the details of the TabularType

For each property entry, the following row is returned

- Property Key - the string key
- Property Value - the stringified version of the property value
- Property Value Type - the type of the property value
getServices

TabularData getServices()
Answer the service state of the system in tabular form

Returns:
the tabular representation of the service state
Throws:
IOException
See Also:
for the details of the TabularType
Each row of the returned table represents a single service. For each service, the following row is returned

- identifier - long
- interfaces - String[]
- bundle - long
- using bundles - long[]

See OSGiService for the precise definition of the CompositeType that defines each row of the table.

getUsingBundles

long[] getUsingBundles(long serviceId)
throws IOException
Answer the list of identifiers of the bundles that use the service

Parameters:
serviceId - the identifier of the service
Returns:
the list of bundle identifiers
Throws:
IOException - if the operation fails
IllegalArgumentException - if the service indicated does not exist
5.7 Selected Compendium Services

These interfaces provide the remote management agent with control APIs for the OSGi compendium services: Configuration Administration, Permission Administration, Permission Manager, User Manager and the Initial Provisioning Service.

5.7.1 Interface ConfigAdminManagerMBean

org.osgi.jmx.compendium Interface ConfigAdminManagerMBean

public interface ConfigAdminManagerMBean

This MBean provides the management interface to the OSGi Configuration Administration Service.

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>void addProperty(String pid, String name, String value, String type)</td>
</tr>
<tr>
<td>Add or update the property for the configuration identified by the supplied pid</td>
</tr>
<tr>
<td>void addProperty(String pid, String location, String name, String value, String type)</td>
</tr>
<tr>
<td>Add or update the property for the configuration identified by the supplied pid and location</td>
</tr>
<tr>
<td>void addPropertyToConfigurations(String filter, String name, String value, String type)</td>
</tr>
<tr>
<td>Add or update the property on all configurations matching the supplied filter</td>
</tr>
<tr>
<td>String createFactoryConfiguration(String factoryPid)</td>
</tr>
<tr>
<td>Create a new configuration instance for the supplied persistent id of the factory, answering the pid of the created configuration</td>
</tr>
<tr>
<td>String createFactoryConfiguration(String factoryPid, String location)</td>
</tr>
<tr>
<td>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</td>
</tr>
<tr>
<td>void delete(String pid)</td>
</tr>
<tr>
<td>void</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>deleteConfigurations(String filter)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delete the configurations matching the filter spec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>deleteProperty(String pid, String key)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delete the property from the configuration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>deleteProperty(String pid, String location, String key)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delete the property from the configuration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>deletePropertyFromConfigurations(String filter, String key)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Remove the property from all configurations matching the supplied filter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>getBundleLocation(String pid)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Answer the bundle location the configuration is bound to</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>getFactoryPid(String pid)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Answer the factory pid if the configuration is a factory configuration, null otherwise.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>getFactoryPid(String pid, String location)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Answer the factory pid if the configuration is a factory configuration, null otherwise.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TabularData</th>
<th>getProperties(String pid)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Answer the credentials of the configuration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TabularData</th>
<th>getProperties(String pid, String location)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Answer the credentials of the configuration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String[][]</th>
<th>listConfigurations(String filter)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Answer the list of PID/Location pairs of the configurations managed by this service</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setBundleLocation(String pid, String location)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set the bundle location the configuration is bound to</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>update(String pid, String location)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
String location,
TabularData properties)
Update the configuration with the supplied properties For each property entry, the following row is supplied

| void update(String pid,
TabularData properties) |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Update the configuration with the supplied properties For each property entry, the following row is supplied</td>
</tr>
</tbody>
</table>

## Method Detail

### addProperty

```java
def addProperty(String pid,
    String name,
    String value,
    String type)
throws IOException
```

Add or update the property for the configuration identified by the supplied pid

**Parameters:**
- `pid` - the persistent id of the configuration
- `name` - the property key to add or update
- `value` - the string encoded property value to add or update
- `type` - the type of the property

**Throws:**
- `IOException` - if the operation fails
- `IllegalArgumentException` - if the filter is invalid

### addProperty

```java
def addProperty(String pid,
    String location,
    String name,
    String value,
    String type)
throws IOException
```

Add or update the property for the configuration identified by the supplied pid and location

**Parameters:**
- `pid` - the persistent id of the configuration
- `location` - the bundle location
- `name` - the property key to add or update
- `value` - the string encoded property value to add or update
- `type` - the type of the property
Throws:
IOException - if the operation fails
IllegalArgumentException - if the filter is invalid

addPropertyToConfigurations

```java
void addPropertyToConfigurations(String filter, String name, String value, String type)
throws IOException
```

Add or update the property on all configurations matching the supplied filter

Parameters:
- filter - the string representation of the Filter
- name - the property key to add or update
- value - the string encoded property value to add or update
- type - the type of the property

Throws:
IOException - if the operation fails
IllegalArgumentException - if the filter is invalid

createFactoryConfiguration

```java
String createFactoryConfiguration(String factoryPid)
throws IOException
```

Create a new configuration instance for the supplied persistent id of the factory, answering the pid of the created configuration

Parameters:
- factoryPid - the persistent id of the factory

Returns:
the pid of the created configuration

Throws:
IOException - if the operation failed

createFactoryConfiguration

```java
String createFactoryConfiguration(String factoryPid, String location)
throws IOException
```

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

Parameters:
- factoryPid - the persistent id of the factory
location -- the bundle location

Returns:
the pid of the created configuration

Throws:
IOException - if the operation failed

delete

void delete(String pid)
  throws IOException
Delete the configuration

Parameters:
pid -- the persistent identifier of the configuration

Throws:
IOException - if the operation fails

delete

void delete(String pid,
  String location)
  throws IOException
Delete the configuration

Parameters:
pid -- the persistent identifier of the configuration
location -- the bundle location

Throws:
IOException - if the operation fails

deleteConfigurations

void deleteConfigurations(String filter)
  throws IOException
Delete the configurations matching the filter spec

Parameters:
filter - the string representation of the Filter

Throws:
IOException - if the operation failed
IllegalArgumentException - if the filter is invalid

deleteProperty

void deleteProperty(String pid,
  String key)
throws IOException
Delete the property from the configuration

**Parameters:**
- **pid** - the persistent identifier of the configuration
- **key** - the property

**Throws:**
- IOException - if the operation fails

### deleteProperty

```java
void deleteProperty(String pid,
                    String location,
                    String key)
```

Delete the property from the configuration

**Parameters:**
- **pid** - the persistent identifier of the configuration
- **location** - the bundle location
- **key** - the property

**Throws:**
- IOException - if the operation fails

### deletePropertyFromConfigurations

```java
void deletePropertyFromConfigurations(String filter,
                                       String key)
```

Remove the property from all configurations matching the supplied filter

**Parameters:**
- **filter** - the string representation of the Filter
- **key** - the property key to be removed

**Throws:**
- IOException - if the operation fails
- IllegalArgumentException - if the filter is invalid

### getBundleLocation

```java
String getBundleLocation(String pid)
```

Answer the bundle location the configuration is bound to

**Parameters:**

pid - the persistent identifier of the configuration

Returns:
the bundle location

Throws:
IOException - if the operation fails

getFactoryPid

String getFactoryPid(String pid)
throws IOException

Answer the factory pid if the configuration is a factory configuration, null otherwise.

Parameters:
pid - the persistent identifier of the configuration

Returns:
the factory pid

Throws:
IOException - if the operation fails

getFactoryPid

String getFactoryPid(String pid, String location)
throws IOException

Answer the factory pid if the configuration is a factory configuration, null otherwise.

Parameters:
pid - the persistent identifier of the configuration
location - the bundle location

Returns:
the factory pid

Throws:
IOException - if the operation fails

getProperties

TabularData getProperties(String pid)
throws IOException

Answer the credentials of the configuration

Parameters:
pid - the persistent identifier of the configuration

Returns:
the table of credentials

Throws:
IOException - if the operation fails

See Also:
for the details of the TabularType
For each property entry, the following row is returned
• Property Key - the string key
• Property Value - the stringified version of the property value
• Property Value Type - the type of the property value

getProperties

TabularData getProperties(String pid, String location)

IOException
Answer the credentials of the configuration

Parameters:
pid -- the persistent identifier of the configuration
location -- the bundle location

Returns:
the table of credentials

Throws:
IOException - if the operation fails

See Also:
for the details of the TabularType
For each property entry, the following row is returned
• Property Key - the string key
• Property Value - the stringified version of the property value
• Property Value Type - the type of the property value

listConfigurations

String[][] listConfigurations(String filter)

IOException
Answer the list of PID/Location pairs of the configurations managed by this service

Parameters:
filter - the string representation of the Filter

Returns:
the list of configuration PID/Location pairs

Throws:
IOException - if the operation failed
IllegalArgumentException - if the filter is invalid

**setBundleLocation**

```java
void setBundleLocation(String pid,
                        String location)
    throws IOException
```

Set the bundle location the configuration is bound to

**Parameters:**

- **pid** - the persistent identifier of the configuration
- **location** - the bundle location

**Throws:**

- IOException - if the operation fails

**update**

```java
void update(String pid,
            TabularData properties)
    throws IOException
```

Update the configuration with the supplied properties For each property entry, the following row is supplied

**Parameters:**

- **pid** - the persistent identifier of the configuration
- **properties** - the table of properties

**Throws:**

- IOException - if the operation fails

**See Also:**

for the details of the TabularType

- Property Key - the string key
- Property Value - the stringified version of the property value
- Property Value Type - the type of the property value

**update**

```java
void update(String pid,
            String location,
            TabularData properties)
    throws IOException
```

Update the configuration with the supplied properties For each property entry, the following row is supplied
Parameters:
pid -- the persistent identifier of the configuration
location -- the bundle location
properties -- the table of properties

Throws:
IOException - if the operation fails

See Also:
for the details of the TabularType

• Property Key - the string key
• Property Value - the stringified version of the property value
• Property Value Type - the type of the property value
5.7.2 Interface PermissionManagerMBean

```
org.osgi.jmx.compendium Interface PermissionManagerMBean
```

```java
public interface PermissionManagerMBean
```

This MBean represents the OSGi Permission Manager Service

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String[] getLocations()</code></td>
<td>Answer the bundle locations that have permissions assigned to them</td>
</tr>
<tr>
<td><code>String[] getPermissions(String location)</code></td>
<td>Answer the list of encoded permissions of the bundle specified by the bundle location</td>
</tr>
<tr>
<td><code>void setDefaultPermissions(String[] encodedPermissions)</code></td>
<td>Set the default permissions assigned to bundle locations that have no assigned permissions</td>
</tr>
<tr>
<td><code>void setPermissions(String location, String[] encodedPermissions)</code></td>
<td>Set the permissions on the bundle specified by the bundle location</td>
</tr>
</tbody>
</table>

### Method Detail

#### getLocations

```java
String[] getLocations() throws IOException
```

Answer the bundle locations that have permissions assigned to them

**Returns:**
the bundle locations

**Throws:**
IOException - if the operation fails

#### getPermissions
String[] getPermissions(String location) throws IOException
Answer the list of encoded permissions of the bundle specified by the bundle location

**Parameters:**
location - location identifying the bundle

**Returns:**
the array of String encoded permissions

**Throws:**
IOException - if the operation fails

**setDefaultPermissions**

void setDefaultPermissions(String[] encodedPermissions) throws IOException
Set the default permissions assigned to bundle locations that have no assigned permissions

**Parameters:**
encodedPermissions - the string encoded permissions

**Throws:**
IOException - if the operation fails

**setPermissions**

void setPermissions(String location, String[] encodedPermissions) throws IOException
Set the permissions on the bundle specified by the bundle location

**Parameters:**
location - the location of the bundle
encodedPermissions - the string encoded permissions to set

**Throws:**
IOException - if the operation fails
5.7.3 Interface ProvisioningMBean

```java
public interface ProvisioningMBean {

    Author: Hal Hildebrand
    Date: Jan 21, 2008
    Time: 10:49:26 AM
    This MBean represents the management interface to the OSGi Initial Provisioning Service

    Method Summary

    void addInformation(String zipURL)
        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.

    void addInformation(TabularData info)
        Adds the key/value pairs contained in info to the Provisioning Information dictionary.

    TabularData getInformation()
        Returns a table representing the Provisioning Information Dictionary.

    void setInformation(TabularData info)
        Replaces the Provisioning Information dictionary with the entries of the supplied table.

    Method Detail

    addInformation

    void addInformation(String zipURL)
        throws IOException
    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.

    Parameters:
```
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 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.

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.

*addInformation*

    void addInformation(TabularData info) throws IOException

    Adds the key/value pairs contained in info to the Provisioning Information dictionary. This method causes the PROVISIONING_UPDATE_COUNT to be incremented.

Parameters:
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.

Throws:
IOException - if the operation fails

See Also:
for the details of the TabularType

    For each entry in the Provisioning Dictionary, the following row is supplied
    • Property Key - the string key
    • Property Value - the stringified version of the property value
    • Property Value Type - the type of the property value

*getInformation*

    TabularData getInformation() 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:
For each entry in the Provisioning Information Dictionary, the following row is supplied:

- Property Key - the string key
- Property Value - the stringified version of the property value
- Property Value Type - the type of the property value

**setInformation**

```java
void setInformation(TabularData info) throws IOException
```

Replaces the Provisioning Information dictionary with the entries of the supplied table. This method causes the PROVISIONING_UPDATE_COUNT to be incremented.

**Parameters:**
info - the new set of Provisioning Information key/value pairs. Any keys are values that are of an invalid type will be silently ignored.

**Throws:**
IOException - if the operation fails

**See Also:**
for the details of the TabularType

For each entry in the table, the following row is supplied:

- Property Key - the string key
- Property Value - the stringified version of the property value
- Property Value Type - the type of the property value
5.7.4 Interface UserManagerMBean

**org.osgi.jmx.compendium Interface UserManagerMBean**

```java
public interface UserManagerMBean
```

**Author:**
Hal Hildebrand Date: Dec 2, 2008 Time: 2:41:26 PM This MBean provides the management interface to the OSGi User Manager Service

**Field Summary**

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>static String[]</td>
<td>AUTHORIZATION</td>
</tr>
<tr>
<td>static String</td>
<td>ENCODED_CREDENTIALS</td>
</tr>
<tr>
<td>static String</td>
<td>ENCODED_ROLE</td>
</tr>
<tr>
<td>static String</td>
<td>ENCODED_USER</td>
</tr>
<tr>
<td>static String[]</td>
<td>GROUP</td>
</tr>
<tr>
<td>static String</td>
<td>GROUP_MEMBERS</td>
</tr>
<tr>
<td>static String</td>
<td>GROUP_REQUIRED_MEMBERS</td>
</tr>
<tr>
<td>static String[]</td>
<td>ROLE</td>
</tr>
<tr>
<td>static String</td>
<td>ROLE_ENCODED_PROPERTIES</td>
</tr>
<tr>
<td>static String</td>
<td>ROLE_NAME</td>
</tr>
<tr>
<td>static String</td>
<td>ROLE_NAMES</td>
</tr>
<tr>
<td>static String</td>
<td>ROLE_TYPE</td>
</tr>
<tr>
<td>static String[]</td>
<td>USER</td>
</tr>
<tr>
<td>static String</td>
<td>USER_NAME</td>
</tr>
</tbody>
</table>

**Method Summary**

```java
void addCredential(String key, byte[] value, ...
```
String username)

Add credentials to a user, associated
with the supplied key

void addCredential(String key, String value, String username)

Add credentials to a user, associated
with the supplied key

boolean addMember(String groupname, String rolename)

Add a role to the group

void addProperty(String key, byte[] value, String rolename)

Add or update a property on a role

void addProperty(String key, String value, String rolename)

Add or update a property on a role

boolean addRequiredMember(String groupname, String rolename)

Add a required member to the
group

void createGroup(String name)

Create a Group

void createUser(String name)

Create a User

CompositeData getAuthorization(String user)

Answer the authorization for the
user name

TabularData getCredentials(String username)

Answer the credentials associated
with a user

CompositeData getGroup(String groupname)

Answer the Group associated with
the groupname

String[] getGroups()

Answer the list of group names

String[] getGroups(String filter)

Answer the list of group names

String[] getImpliedRoles(String username)

Answer the list of implied roles for
a user

String[] getMembers(String groupname)

Answer the the user names which
are members of the group

TabularData getProperties(String rolename)
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String[] getRequiredMembers(String groupname)</td>
<td>Answer the list of user names which are required members of this group</td>
</tr>
<tr>
<td>CompositeData getRole(String name)</td>
<td>Answer the role associated with a name</td>
</tr>
<tr>
<td>String[] getRoles()</td>
<td>Answer the list of role names in the User Admin database</td>
</tr>
<tr>
<td>String[] getRoles(String filter)</td>
<td>Answer the list of role names which match the supplied filter</td>
</tr>
<tr>
<td>CompositeData getUser(String username)</td>
<td>Answer the User associated with the username</td>
</tr>
<tr>
<td>String getUser(String key, String value)</td>
<td>Answer the user name with the given property key-value pair from the User Admin service database.</td>
</tr>
<tr>
<td>String[] getUsers()</td>
<td>Answer the list of user names in the User Admin database</td>
</tr>
<tr>
<td>String[] getUsers(String filter)</td>
<td>Answer the list of user names in the User Admin database</td>
</tr>
<tr>
<td>void removeCredential(String key, String username)</td>
<td>Remove the credentials associated with the key for the user</td>
</tr>
<tr>
<td>boolean removeMember(String groupname, String rolename)</td>
<td>Remove a role from the group</td>
</tr>
<tr>
<td>void removeProperty(String key, String rolename)</td>
<td>Remove a property from a role</td>
</tr>
<tr>
<td>boolean removeRole(String name)</td>
<td>Remove the Role associated with the name</td>
</tr>
</tbody>
</table>

**Field Detail**

**ROLE_NAME**

static final String ROLE_NAME

See Also:
Constant Field Values

ROLE_TYPE

static final String ROLE_TYPE
See Also:
Constant Field Values

ROLE_ENCODED_PROPERTIES

static final String ROLE_ENCODED_PROPERTIES
See Also:
Constant Field Values

ENCODED_USER

static final String ENCODED_USER
See Also:
Constant Field Values

GROUP_MEMBERS

static final String GROUP_MEMBERS
See Also:
Constant Field Values

GROUP_REQUIRED_MEMBERS

static final String GROUP_REQUIRED_MEMBERS
See Also:
Constant Field Values

USER_NAME

static final String USER_NAME
See Also:
Constant Field Values

ROLE_NAMES

static final String ROLE_NAMES
See Also:
Constant Field Values

ENCODED_ROLE
static final String ENCODED_ROLE
See Also:
Constant Field Values

ENCODED_CREDENTIALS

static final String ENCODED_CREDENTIALS
See Also:
Constant Field Values

AUTHORIZATION

static final String[] AUTHORIZATION

USER

static final String[] USER

ROLE

static final String[] ROLE

GROUP

static final String[] GROUP

Method Detail

addCredential

void addCredential(String key,
                   byte[] value,
                   String username)
       throws IOException

Add credentials to a user, associated with the supplied key

Parameters:
key -
value -
username -

Throws:
IOException - if the operation fails
IllegalArgumentException - if the username is not a User
addCredential

```java
void addCredential(String key,
                   String value,
                   String username)
throws IOException
```

Add credentials to a user, associated with the supplied key.

**Parameters:**
- `key` -
- `value` -
- `username` -

**Throws:**
- `IOException` - if the operation fails
- `IllegalArgumentException` - if the username is not a User

addMember

```java
boolean addMember(String groupname,
                   String rolename)
throws IOException
```

Add a role to the group.

**Parameters:**
- `groupname` -
- `rolename` -

**Returns:**
- `true` if the role was added to the group

**Throws:**
- `IOException` - if the operation fails

addProperty

```java
void addProperty(String key,
                  String value,
                  String rolename)
throws IOException
```

Add or update a property on a role.

**Parameters:**
- `key` - the property key
- `value` - the String property value
- `rolename` - the role name

**Throws:**
- `IOException` - if the operation fails
addProperty

    void addProperty(String key, 
                     byte[] value, 
                     String rolename) 
    throws IOException

Add or update a property on a role

Parameters:
    key -- the property key
    value -- the byte[] property value
    rolename -- the role name

Throws:
    IOException - if the operation fails

addRequiredMember

    boolean addRequiredMember(String groupname, 
                              String rolename) 
    throws IOException

Add a required member to the group

Parameters:
    groupname -
    rolename -

Returns:
true if the role was added to the group

Throws:
IOException - if the operation fails

createUser

    void createUser(String name) 
    throws IOException

Create a User

Parameters:
    name -- the user to create

Throws:
IOException - if the operation fails

createGroup

    void createGroup(String name) 
    throws IOException

Create a Group
Parameters:
name - the group to create

Throws:
IOException - if the operation fails

getAuthorization

CompositeData getAuthorization(String user) throws

IOException
Answer the authorization for the user name

Parameters:
user -

Returns:
the Authorization

Throws:
IOException - if the operation fails
IllegalArgumentException - if the username is not a User

See Also:
for the details of the CompositeType

g getCreden cials

TabularData getCredentials(String username) throws

IOException
Answer the credentials associated with a user

Parameters:
username -

Returns:
the credentials associated with the user

Throws:
IOException - if the operation fails
IllegalArgumentException - if the username is not a User

See Also:
for the details of the TabularType
getGroup

CompositeData `getGroup(String groupname)`

Answer the Group associated with the groupname

**Parameters:**
groupname -

**Returns:**
the Group

**Throws:**
IOException - if the operation fails
IllegalArgumentException - if the groupname is not a Group

**See Also:**
for the details of the CompositeType

getGroups

String[] `getGroups()`

Answer the list of group names

**Returns:**
the list of group names

**Throws:**
IOException - if the operation fails

getGroups

String[] `getGroups(String filter)`

Answer the list of group names

**Parameters:**
filter -- the filter to apply

**Returns:**
the list of group names

**Throws:**
IOException - if the operation fails

getImpliedRoles
String[] getImpliedRoles(String username) throws IOException

Answer the list of implied roles for a user

Parameters:
username -

Returns:
the list of role names

Throws:
IOException - if the operation fails
IllegalArgumentException - if the username is not a User

getMembers

String[] getMembers(String groupname) throws IOException

Answer the the user names which are members of the group

Parameters:
groupName -

Returns:
the list of user names

Throws:
IOException - if the operation fails
IllegalArgumentException - if the groupname is not a group

getProperties

TabularData getProperties(String rolename) throws IOException

Answer the credentials associated with a role

Parameters:
rolename -

Returns:
the credentials associated with the role

Throws:
IOException - if the operation fails
See Also:
for the details of the TabularType

getRequiredMembers
String[] getRequiredMembers(String groupname) throws IOException

Answer the list of user names which are required members of this group

Parameters:
  groupname -

Returns:
  the list of user names

Throws:
  IOException - if the operation fails
  IllegalArgumentException - if the groupname is not a group

getRole

CompositeData getRole(String name) throws IOException

Answer the role associated with a name

Parameters:
  name -

Returns:
  the Role

Throws:
  IOException - if the operation fails

See Also:
  for the details of the CompositeType

getRoles

String[] getRoles() throws IOException

Answer the list of role names in the User Admin database

Returns:
  the list of role names

Throws:
  IOException - if the operation fails

getRoles

String[] getRoles(String filter) throws IOException
Answer the list of role names which match the supplied filter

**Parameters:**
- `filter` - the string representation of the Filter

**Returns:**
- the list the role names

**Throws:**
- IOException - if the operation fails

**getUser**

```java
CompositeData getUser(String username)
```

Answer the User associated with the username

**Parameters:**
- `username` -

**Returns:**
- the User

**Throws:**
- IOException - if the operation fails
- IllegalArgumentException - if the username is not a User

**See Also:**
- for the details of the CompositeType

**getUser**

```java
String getUser(String key, String value)
```

Answer the user name with the given property key-value pair from the User Admin service database.

**Parameters:**
- `key` -- the key to compare
- `value` -- the value to compare

**Returns:**
- the User

**Throws:**
- IOException - if the operation fails

**getUsers**
String[] getUsers() throws IOException
Answer the list of user names in the User Admin database

**Returns:**
the list of user names

**Throws:**
IOException - if the operation fails

---

String[] getUsers(String filter) throws IOException
Answer the list of user names in the User Admin database

**Parameters:**
filter -- the filter to apply

**Returns:**
the list of user names

**Throws:**
IOException - if the operation fails

---

void removeCredential(String key, String username) throws IOException
Remove the credentials associated with the key for the user

**Parameters:**
key -
username -

**Throws:**
IOException - if the operation fails
IllegalArgumentException - if the username is not a User

---

boolean removeMember(String groupname, String rolename) throws IOException
Remove a role from the group

**Parameters:**
groupname -
rolename -

Returns:
true if the role was removed from the group

Throws:
IOException - if the operation fails
IllegalArgumentException - if the groupname is not a Group

removeProperty

void removeProperty(String key,
                     String rolename)

throws IOException

Remove a property from a role

Parameters:
key -
rolename -

Throws:
IOException - if the operation fails

removeRole

boolean removeRole(String name)

throws IOException

Remove the Role associated with the name

Parameters:
name -

Returns:
true if the remove succeeded

Throws:
IOException - if the operation fails

6 Considered Alternatives

This section explores various mechanisms for exposing the OSGi framework API into JMX and documents the potential shortcomings of each.

6.1 Direct Translation of the OSGi Framework APIs

A straightforward approach is to simply replicate the OSGi framework APIs, directly translating concrete classes into interfaces and augmenting existing interfaces to transform them into a JMX compliant system. This approach provides a very RMI like interface to underlying framework APIs, in that the exposed objects have a direct one to
one relationship with the framework artifacts which replicates the underlying APIs as closely as possible. The problem with this approach is that the underlying OSGi framework APIs are designed for in process, direct manipulation and are not abstracted and designed to facilitate the remote management of the framework. Operations which are completely natural when performed in the same process become cumbersome and impractical when viewed from the perspective of a remote management agent.

### 6.2 Automatic JMX Translation of OSGi Framework and Services

There are a number of very nice systems which provide various degrees of transparently publishing existing services and frameworks into JMX. These systems have the advantage of not requiring any changes to the underlying systems being exposed through the JMX framework, however they share the same disadvantages of a direct translation, discussed in the previous section, in that the systems were not designed with remote management in mind and nothing these automatic translation systems can do will change that.

### 6.3 JMX Translation of Management Technology Neutral Refactoring

The idea here is that there would be value in coming up with a technology neutral facade which could then be consumed by JMX. There is potentially a lot of value in providing a technology “neutral” management API in that this could be reused in other technologies as well as JMX. However, the artifacts of a JMX compliant interface are largely Java interfaces and perhaps some concrete classes for interchange. This means that implementations can make use of techniques such as the javax.management.StandardMBean[5] which allows straightforward implementation of the interfaces with JMX standard MBean techniques without polluting the package namespace with the JMX implementation. Consequently, the interfaces defined for JMX management of OSGi do not have any JMX bleed through which would prevent it from being in other management frameworks.

### 7 Security Considerations

The management interfaces in this specification are designed for use with the JMX framework. JMX has its own security permission framework as well as specification of remote authentication and authorization. Consequently, all security considerations are delegated to the enclosing JMX framework which hosts these interfaces.

### 8 Document Support

#### 8.1 References


[3]. Java Management Extensions (JMX) Technology Overview
http://java.sun.com/j2se/1.5.0/docs/guide/jmx/overview/JMXoverviewTOC.html
[4]. JavaTM Management Extensions (JMXTM) API Specification
   http://java.sun.com/j2se/1.5.0/docs/guide/jmx/spec.html

[5]. Javax.management.StandardMBean

[6]. Using JConsole to Monitor Applications

8.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Hal Hildebrand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Oracle</td>
</tr>
<tr>
<td>Address</td>
<td>500 Oracle Parkway, M/S 2op946, Redwood City, CA 94065</td>
</tr>
<tr>
<td>Voice</td>
<td>+1 650 563 9646</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:hal.hildebrand@oracle.com">hal.hildebrand@oracle.com</a></td>
</tr>
</tbody>
</table>

8.3 Acronyms and Abbreviations

8.4 End of Document
Abstract

This document describes how JNDI can be integrated into the OSGi environment. The usage of JNDI in Java Enterprise Edition application servers is given the most focus. One main consideration should be that Java EE applications that comply with the standard should work as written in an application server running on OSGi. This document presents an approach towards supporting JNDI contexts in an OSGi server. This document will also include possible approaches toward supporting the usage of JNDI that is required in a Java EE server.

Please Note: While the requirements section of this RFC is complete, portions of the design specification are still under discussion. The design section of this document should be considered a “work in progress”.

Copyright © Oracle 2009

This contribution is made to the OSGi Alliance as MEMBER LICENSED MATERIALS pursuant to the terms of the OSGi Member Agreement and specifically the license rights and warranty disclaimers as set forth in Sections 3.2 and 12.1, respectively.

All company, brand and product names contained within this document may be trademarks that are the sole property of the respective owners. The above notice must be included on all copies of this document that are made.
0 Document Information

0.1 Table of Contents

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

1 Introduction.........................................................................................3

2 Application Domain...........................................................................3

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

4 Requirements.....................................................................................4

5 Technical Solution..............................................................................4

6 Considered Alternatives.....................................................................4

7 Security Considerations....................................................................5

8 Document Support...............................................................................5
  8.1 References..................................................................................5
  8.2 Author’s Address..........................................................................5
  8.3 Acronyms and Abbreviations......................................................5
  8.4 End of Document..........................................................................5

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 3.1.

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  | 09 30 2008 | Initial Draft  
John Wells, Oracle john.wells@oracle.com                                                                                           |
| Draft1   | 10 07 2008 | Initial Draft Content  
Robert W. Nettleton, bob.nettleton@oracle.com                                                                                     |
| Draft2   | 10 13 2008 | Modified draft, incorporated some internal feedback from Jeff Trent  
Robert W. Nettleton, bob.nettleton@oracle.com                                                                                     |
| Version 0.2 | 12 3 2008 | Version 0.2 of draft. Added requirement to support lookups of OSGi services. Incorporated some feedback from Graham Charters, including his suggestion for an “osgi” services URL.  
Robert W. Nettleton, bob.nettleton@oracle.com                                                                                     |
| Version 0.3 | 1 9 2009   | Version 0.3 of draft. Added a proposal for a possible solution of the issues around using URL context factories in OSGi/JNDI  
Robert W. Nettleton, bob.nettleton@oracle.com                                                                                     |
| Version 0.4 | 1 22 2009  | Version 0.4 of draft. Added more details on how the factory manager could locate URL context factory implementations. Additional content added concerning the differences between URL Service Handlers and URL Context Factories.  
Robert W. Nettleton, bob.nettleton@oracle.com                                                                                     |
| Version 0.5 | 2 5 2009   | Version 0.5 of draft. Incorporated feedback from EEG group as a result of the meeting on 1 30 2009.  
Incorporated feedback from Ben Hale from SpringSource on JNDI lookups of OSGi services.  
Robert W. Nettleton, bob.nettleton@oracle.com                                                                                     |
| Version 0.6 | 3 3 2009   | Version 0.6 of draft. Incorporated feedback from EEG meeting on 2 27 2009.  
Robert W. Nettleton, bob.nettleton@oracle.com                                                                                     |
| Version 0.7 | 3 5 2009   | Version 0.7 of draft. Early Release Draft  
Robert W. Nettleton, bob.nettleton@oracle.com                                                                                     |
1 Introduction

JNDI (the “Java Naming and Directory Interface”) is a popular registry technology in the enterprise space. OSGi has its own object registry model. This paper shall attempt to define the points whereby the JNDI registry and the OSGi registry can be integrated.

This document will suggest an approach towards integrating JNDI into an OSGi-enabled application. Since JNDI can be considered a technology on its own, a portion of this solution will allow for a pure JNDI provider to integrate into an OSGi-based server.

In addition to the JNDI itself, the Java EE (“Java Enterprise Edition”) specification requires JNDI contexts to be handled in a special way, depending upon the configuration of a deployed application. In particular, application namespaces are expected to be treated separately in Java EE applications. The “environment naming context” (also known as “java:comp/env”) is also required to be supported in a Java EE server. An application server is required to provide unique environment naming contexts for each component in an application. An example of a component in this case would be a servlet or an EJB.

2 Application Domain

Enterprise code using Java EE depends heavily on the use of JNDI to register and find objects. The specification details elements of JNDI that must be supported in order to properly deploy Java EE applications.

3 Problem Description

3.1 Basic JNDI Support

Java EE Applications deployed into an OSGi-based server need to be able to access a naming service in order to locate services. In addition, there may be other applications that don’t conform to the Java EE specification, but may require a naming service to properly locate services. The OSGi framework needs to provide an extensible mechanism for publishing and finding both JNDI InitialContextFactory and Context instances.
3.1.1 Use Case 1: Default Context Creation

JNDI clients may choose to rely on a container environment or configuration to determine the JNDI custom implementation to use when using a naming service. This environment can be provided by a container, such as an application server, or can also be configured as a set of system properties in the JVM.

The following is an example of creating a JNDI InitialContext with the default configuration:

```java
InitialContext context = new InitialContext();
DataSource dataSource = (DataSource)context.lookup("accountingDataSourceName");
```

3.1.2 Use Case 2: User-specified JNDI implementation

Another common use case with JNDI Clients involves the client specifying the custom InitialContextFactory to use in order to create the InitialContext. The following code snippet demonstrates this use case:

```java
Hashtable environment = new Hashtable();

environment.put(Context.INITIAL_CONTEXT_FACTORY, 
"com.demo.DemoInitialContextFactory");

InitialContext context = new InitialContext(environment);
```

In this example, a client configures the JNDI framework to locate and create an instance of the "DemoInitialContextFactory". The JNDI framework is then expected to create a new `javax.naming.Context` instance to return to the client.

Both of these use cases must be supported for JNDI clients running within an OSGi container.

3.1.3 Use Case 3: URL Context Factory Support

The JNDI framework defines a mechanism by which a JNDI service provider can resolve lookups for custom Uniform Resource Locator (URL) forms.

The following code example demonstrates the usage of a URL in a JNDI lookup:

```java
InitialContext initialContext = new InitialContext();

// retrieve an object using a URL based on the “java” scheme
HelloBean hello = (HelloBean)initialContext.lookup("java:comp/env/Hello");

// another example, retrieve an object using the custom “util” scheme
ThreadPoolManager manager =
    (ThreadPoolManager)initialContext.lookup("util://ThreadPoolManager");
```

These code examples demonstrate the most common usage pattern for URL based lookups. Many of these types of lookups will be required by customers regardless of the `InitialContextFactory` used.
In a pure JNDI (non-OSGi) setting, service providers can make an implementation of `javax.naming.spi.ObjectFactory` available to create a JNDI context using a URL as an input. This `ObjectFactory` implementation is known as a “URL Context Factory”, since the factory creates a context given a URL string. The details of how a service provider makes this factory available can be found at the following link:


A URL Context Factory allows a service provider to extend the default behavior of the JNDI runtime without having to provide an `InitialContextFactory`. Once available to the JNDI runtime, a URL Context factory can be used with any initial context implementation, since the `javax.naming.InitialContext` class provides the support for locating factories based on a given URL scheme.

In a pure JNDI setting, the service provider must follow a well-defined naming convention for the URL context factory, and then make this factory's package available via a system property used by the JNDI framework.

Support for JNDI applications in OSGi will require the handling of URL context factories. The default JNDI support for this type of factory will be problematic in OSGi, given the reliance on system properties. This issue is discussed in greater detail in another section of this document.

### 3.2 Classloading

JNDI was designed and added to the Java platform before OSGi was prevalent, and it does not deal well with the OSGi class loading architecture. In particular, the JNDI code running inside the Java VM by default expects that key JNDI classes, such as context factories and object factories, are loaded using the system class loader. There is no provision in the JNDI specification to plug in a different classloading system such as OSGi.

In most cases, JNDI supports APIs that allow a user to change the way that classes are loaded, so that it is possible to take advantage of the OSGi environment. However, for various detailed technical reasons in the design and implementation of the JNDI APIs, there are reasons why these APIs do not solve the entire problem. In particular, once builder classes are installed with the `NamingManager`, there is no mechanism to un-set or re-set the builder classes. This causes the JVM to hold onto a reference to any installed builder, as well as the classes that represent that builder. This issue is discussed in greater detail in another section of this document.

### 3.3 Java Enterprise Edition Support

#### 3.3.1 Application Namespace

While JNDI exists as a specification on its own, the Java EE application server environment imposes an additional set of requirements on a JNDI provider. The default JNDI provider in an application server is usually available to deployed applications without additional configuration.

A typical example of this usage would be a Java EE Servlet locating a JDBC datasource, using the following code snippet:

```java
InitialContext context = new InitialContext();

DataSource dataSource = (DataSource)context.lookup("accounting_ds");
```
This code snippet demonstrates that JNDI is heavily integrated into a Java EE application server. In a pure JNDI scenario, the client would typically need to provide a list of properties to configure the `InitialContextFactory` to be used. In the example above, the application server handles the task of providing a `javax.naming.Context` instance that can satisfy the requests for a given application. Application servers can manage separate context instances for each application deployed. The Java EE specification requires that each deployed application have its own unique namespace. Vendors can implement this by creating unique `javax.naming.Context` instances for each application.

### 3.3.2 Environment Naming Context

Java EE application servers are also required to support logical references to different types of resources. Some examples of a Java EE resource would include: JDBC Datasources, EJB references, JMS Topics/Queues, etc. While it is always possible for a client of JNDI to directly lookup an object under the name it was originally bound, there are cases where this is not desirable. In the Datasource example above, it may be more convenient to create a logical name to a given datasource, then allow application deployment descriptors to map this logical name to a binding in the JNDI context. This approach supports more flexibility at deployment-time, since the datasource used in this example may be changed without code modification.

Java EE application servers are required to support logical references, also known as “resource references”, in a very specific way. Each component container (Servlet, EJB) must provide a namespace of logical names that is unique for each deployed component. This namespace is known as the “environment naming context”. This context is typically available under the “java:comp/env” namespace.

Another point to consider is that Java EE servers must manage JNDI contexts depending upon the environment being used. The environment naming context is required to be a read-only context, and the Java EE specification requires that an application server enforce this restriction.

In addition to the problem listed in Section 3.1, OSGi-based servers will need to support namespace boundaries at the Java EE application level. Much of this support can be provided by the application server, but the basic integration points need to be provided at the framework level.

### 3.4 OSGi Service Integration

#### 3.4.1 OSGi Service Access

In addition to the Java EE support described above, it may be desirable to allow JNDI clients to access OSGi services via the JNDI context. This would allow clients to migrate to an OSGi services-based model with minimal code changes. The client would use the `Context.lookup()` method to locate a service, and then interact with that service in the expected way.

In order to support this feature, the OSGi framework should provide a JNDI Context implementation that can resolve lookup requests for the "osgi" namespace.

*This access is defined to be read-only, in that only OSGi service access through the JNDI context is permitted.*

The following is an example of this type of lookup:

```java
InitialContext context = new InitialContext();
```
ExampleService = (ExampleService)context.lookup("osgi:services/com.example.ExampleService");

This example utilizes an “osgi”-based URL form that JNDI clients can pass to the lookup() method of the JNDI Context.

### 3.4.2 Handling of OSGi ServiceReferences

When OSGi clients access services, either by using the BundleContext API or the ServiceTracker, the usage pattern typically involves the following operations:

1. Obtain reference to service (using `BundleContext.getServiceReference()`)  
2. Invoke operations on service  
3. Release reference to service (using `BundleContext.ungetService()`)  

The `javax.naming.Context` interface does not support such a model directly. In typical JNDI scenarios, clients access an object using a JNDI name, and then interact with the object. There is no support in the JNDI programming model for informing the provider that a given object is no longer needed by the client.

A JNDI provider that handles requests for OSGi services will need to implement some approach for handling the lifecycles of the OSGi services made available via JNDI.

### 3.4.1 BundleContext Handling

A JNDI provider that handles lookup requests for OSGi services will need to make service requests on behalf of the bundle that the JNDI client executes in. This generally means that the client's BundleContext instance must be used to invoke the `getService()` call. The client's bundle context must be used in this scenario for the following reasons:

1. **Classloading** - The JNDI provider needs to insure that the proper classloader is always used to load the classes for a given service. For example: if bundle A attempts a lookup of service "my.test.Service" that happens to be deployed in bundle B, there is always a chance that bundle A will have an "Import-Package" statement that pulls in the "my.test" package from another bundle, C in this case. In this example, when bundle A attempts to cast the object returned from the lookup to the service type would fail, since an object of the interface type loaded from bundle B is returned, but the cast expects the interface type to be loaded by bundle C's loader. The OSGi Specification requires that BundleContext implementations of `getServiceReference` and `getServiceReferences` filter out any services with "incompatible" types. When a caller uses the BundleContext API directly, the `ClassCastException` will not occur, since an incompatible service won't be returned. The JNDI provider in this case must have access to the client's BundleContext.

2. **Service Tracking** – OSGi services returned to JNDI clients must be tracked appropriately in order to allow the OSGi Framework to properly manage service dependencies. Using the JNDI client's BundleContext to acquire and release services is the only way to achieve this.
1 Requirements

1. A JNDI application (defined as an OSGi bundle using JNDI, or a Java EE application that is deployed on an OSGi-enabled container) MUST be able to use the InitialContext class to create a Context object as long as the InitialContextFactory class is exported from an OSGi bundle, or resides on the system class path.

2. When a Referenceable JNDI object is retrieved from JNDI, code running in any OSGi bundle MUST be able to successfully retrieve the object, as long as the object’s ObjectFactory or DirObjectFactory is exported by another bundle, or resides on the system class path.

3. There MUST be a way to export an ObjectFactory from an OSGi bundle so that it may be used as a “URL Context” (also known as a “URL Context Factory”) to allow JNDI access via URL.

4. There MAY be additional support for URL context factories to simplify the task of creating the URL context factory.

5. There SHOULD be a way to add an ObjectFactory, DirObjectFactory, StateFactory, or DirStateFactory to the list of factories to be searched by default by JNDI by registering a service with the OSGi service registry, in addition to the methods already supported by JNDI.

6. Any OSGi bundle MUST be able to look up and fully utilize a JNDI Context object bound in to the Service Registry.

7. There SHOULD be a configurable mechanism to make it easy to bind JNDI Context objects to the Service Registry.

8. There MUST be a standard that describes the mechanism that JNDI will use to search for environment properties when running in an OSGi environment. The standard must describe how JNDI environment properties, especially those stored in resource files, are loaded, and how that differs from a non-OSGi environment.

9. There MUST be a mechanism for locating javax.naming.Context instances that support a given Java EE application. Any proposed solution should include a flexible OSGi service-based approach towards publishing JNDI contexts that support a given application, as well as provide a standard way for components to locate and use these JNDI providers.

10. There MUST be a way to access OSGi services via a JNDI context. This feature will require a way to specify the JNDI name that a service may be published under, as well as support for managing the lifecycle of the OSGi service.

11. Once a client accesses an OSGi service via JNDI, the OSGi/JNDI integration solution MUST monitor the usage of the service reference, in order to properly maintain the lifecycle of the service.
1 Technical Solution

1.1 Overview

In general, the components in an OSGi-enabled application server will need to locate and publish Java EE-related services via JNDI. This means that bundles will need to be able to locate JNDI contexts in a standard way, and it will also be desirable to have bundles publish their `javax.naming.Context` implementations for other bundles to consume.

An important goal of this integration will be to support Java EE applications that are running in an OSGi-based container. These Java EE applications should not have to be modified in order to run in a Java EE server that just happens to be using OSGi infrastructure. This presents a challenge to application server vendors, since typical application servers handle issues such as classloading and service location in a different fashion than OSGi.

Note: While the concept of an application is essential to Java EE development, this specification does not require that an OSGi-enabled server implement that concept by using a one-to-one mapping of applications to bundles. This partitioning may be accomplished in any number of ways. The most significant requirement in this area revolves around the notion that a Java EE application must be kept separate from other deployed EE applications.

1.1.1 InitialContextFactory Creation

Section 3.3.1 lists a small code snippet, that is repeated here:

```java
InitialContext context = new InitialContext();
```

In many cases, JNDI clients can create a JNDI context using the default constructor listed above. This requires the application server to provide a context implementation that can represent the current Java EE application. The JNDI framework relies on environment properties (either passed in at construction time, or set by a component container) to determine which `javax.naming.InitialContextFactory` to use to construct the context.

Another common use case with JNDI Clients involves the client specifying the custom `InitialContextFactory` to use in order to create the InitialContext. The following code snippet demonstrates this use case:

```java
Hashtable environment = new Hashtable();

environment.put(Context.INITIAL_CONTEXT_FACTORY, "com.demo.DemoInitialContextFactory");

InitialContext context = new InitialContext(environment);
```

In this example, a client configures the JNDI framework to locate and create an instance of the “DemoInitialContextFactory”. The JNDI framework is then expected to create a new `javax.naming.Context` instance to return to the client.
The JNDI framework also provides an additional integration point that allows a component to control how JNDI contexts are created. A component can provide an implementation of:

```java
javax.naming.spi.InitialContextFactoryBuilder,
```

which is a builder for `InitialContextFactory` instances. A component can provide a builder implementation to the JNDI framework by calling the following API method:

```java
javax.naming.spi.NamingManager.setInitialContextFactoryBuilder(InitialContextFactoryBuilder)
```

This method can only be called once per Virtual Machine run, and cannot be overwritten once set. The component must also have the proper security permissions for this call to succeed. Once this method has been called, the JNDI framework will delegate any context creation requests to this builder implementation. This allows a JNDI provider to have more fine-grained control over the creation of contexts, and provides a single entry point into the naming subsystem. The sequence diagram in Figure 5-1 gives an overview of the interaction involved during an attempt to create a new InitialContext. Except for “Client”, all classes listed in this diagram are part of the standard JNDI framework, and exist in the “javax.naming.*” packages.

![Figure 5-1 Sequence Diagram of JNDI Context Creation with Builder installed](image)

### 1.1.2 ObjectFactory Creation

The JNDI provides a framework for customizing the binding that is returned to a caller. The most common example would be a call to `javax.naming.Context.lookup(String)`. For example: a JNDI client creates a `Context` instance, and attempts to lookup a reference to a `Printer` object. The following code snippet demonstrates this type of lookup:

```java
InitialContext initialContext = new InitialContext();
```
In the simplest case, the JNDI service provider may bind the Printer object directly to the name “test_printer”. There may, however, be a case for binding a `javax.naming.Reference` object instead. Binding a Reference object allows for a level of indirection when this name is resolved. At lookup time, the JNDI can use the Reference to create a Printer object to return to the caller. This could allow for more flexible object creation policies, such as a “lazy” object creation scheme for objects that require large amounts of resources.

This object creation mechanism is provided by the `javax.naming.spi.ObjectFactory` interface. JNDI service providers that need to allow for this type of reference resolution implement the `ObjectFactory` interface. While resolving a lookup() request, a JNDI context is required to test the object's type to determine if the object is a `javax.naming.Reference`. If this test passes, the context implementation is required to call `javax.naming.spi.NamingManager.getObjectInstance(Object, Name, Context, Hashtable)` in order to resolve the reference. The `NamingManager` will then go through a series of attempts to find a factory to build the object from the Reference. The steps for this resolution are defined in the `NamingManager`'s javadoc, and in many cases depend upon the `Context.OBJECT_FACTORIES` system property. The order of object factories listed in this system property can determine which factory is consulted first in an attempt to build the object.

The JNDI's default mechanism for finding `ObjectFactory` instances is largely dependent upon the value of the system property mentioned above. For this reason, any JNDI integration with OSGi will need to take a different approach in order to support resolving `javax.naming.Reference` instances.

The JNDI provides an integration point with respect to finding `ObjectFactory` instances that is very similar to the `InitialContextFactoryBuilder` approach listed in Section 5.1.1. A component can provide an implementation of:

`javax.naming.spi.ObjectFactoryBuilder`,

which is a builder for `ObjectFactory` instances. A component can provide a builder implementation to the JNDI framework by calling the following API method:

`javax.naming.spi.NamingManager.setObjectFactoryBuilder(ObjectFactoryBuilder)`

This method can only be called once per Virtual Machine run, and cannot be overwritten once set. The component must also have the proper security permissions for this call to succeed. Once this method has been called, the JNDI framework will delegate any attempts to resolve a `javax.naming.Reference` with an `javax.naming.spi.ObjectFactory` to this builder implementation. This allows a JNDI provider to have more fine-grained control over the creation of object factories, and also more control over the manner in which references are resolved into objects. The sequence diagram in Figure 5-2 gives an overview of the interaction involved during an attempt to lookup a name that is bound to a Reference. Except for “Client”, all classes listed in this diagram are part of the standard JNDI framework, and exist in the “javax.naming.” packages.
1.2 Factory Manager bundle

This document proposes using a special bundle, known as the “factory-manager” bundle, to handle the integration with JNDI and OSGi. This bundle will be considered a default part of the OSGi framework, and must be installed for the integration to succeed. During bundle activation, this “factory-manager” bundle will provide implementations of InitialContextFactoryBuilder and ObjectFactoryBuilder to the JNDI NamingManager. These builder implementations will interact with the OSGi service registry in order to discover new InitialContextFactory, InitialContextFactoryBuilder, ObjectFactory, ObjectFactoryBuilder, and Context implementations. This will allow the bundle to dynamically provide JNDI access to any applications and bundles that publish such information on the registry. This bundle will provide a default implementation of the InitialContextFactoryBuilder and ObjectFactoryBuilder in order to allow existing context factories to be placed on the classpath and used “as-is”. For a more flexible approach, a bundle can publish an OSGi service that implements the InitialContextFactoryBuilder interface, the InitialContextFactory interface, or both to handle context factory creation. A bundle will also be able to publish an OSGi service that implements the ObjectFactory interface or the ObjectFactoryBuilder interface, in order to support more flexible reference resolution. The “factory-manager” bundle will interact with the OSGi framework to detect the publishing of services that implement these interfaces, as well as the removal of these services from the OSGi registry. The method of tracking services is considered an implementation detail, and implementations of this specification are free to use any method that is most appropriate (ServiceListener, Service Tracker, etc).

1.2.1 InitialContextFactoryBuilder Integration

The following diagram details how the “factory-manager” bundle can integrate with the JNDI framework to customize the creation of InitialContextFactory instances, which ultimately allow an implementer to customize the creation of javax.naming.Context instances.
The diagram above shows one possible way to integrate with JNDI using the standard NamingManager. Some differences with the first diagram (5-1) are worth mentioning:

1. **OSGIInitialContextFactoryBuilder** – This class is an implementation of `javax.naming.spi.InitialContextFactoryBuilder`. Upon activation of the “factory-manager” bundle, this builder is created and registered with the NamingManager. After this registration, all attempts to create a JNDI InitialContext will be routed through this builder. The builder is free to create new factory implementations, delegate to existing factories, or build proxy implementations as needed. This class can be considered the main entry point between the JNDI framework and and OSGi implementation of JNDI. The proposed implementation will delegate the work of managing these OSGi services to the FactoryManager.

2. **FactoryManager** – This class interacts with OSGi in order to detect any OSGi services that are published that conform to the InitialContextFactoryBuilder interface. This class can utilize OSGi Framework APIs to detect the published services available, as well as track the registration/de-registration of services during runtime. The method used by this component to track services is not specified, and is left as an implementation detail. The presence of this particular class is not necessarily required to implement this bundle. This class is in place to represent the code that will integrate with OSGi in order to detect available JNDI providers. Figure 5-5 below describes how the FactoryManager can register itself with the OSGi service registry in order to locate InitialContextFactoryBuilder and ObjectFactory services.

3. **InitialContextFactoryBuilder** – In this diagram, this class can represent any implementation of the factory builder interface that is available via OSGi services. This implementation can be provided by the “factory manager” bundle itself, or can be one of an arbitrary number of services published in the OSGi service registry.

1.2.1.1 Behavior of FactoryManager with Context Factories

The FactoryManager class is used in the above diagrams to represent the internal implementation of the factory manager bundle. While the class itself is not required by implementors of this specification, there are some requirements on how this bundle locates and returns JNDI services.

The JNDI framework allows customization at several levels. A service provider typically implements `javax.naming.spi.InitialContextFactory` in order to provide a custom JNDI context factory. While this is the common case, it is also possible that a service provider may wish to provide an `InitialContextFactoryBuilder`, in order to support more fine-grained control over the creation of context factories.

This document proposes that the “factory manager” bundle support both interfaces using the following mechanism:

When the `InitialContextFactoryBuilder` instance installed by the “factory manager” bundle is invoked by the framework with a request to provide an `InitialContextFactory` instance, it should use the following search order:

1. If a specific `InitialContextFactory` class is specified by the client, the “factory manager” must attempt to locate a service that supports an interface defined by the classname of the custom factory. If this service is found, it should be returned to the client. If a service is not found to be registered under this factory class name, this should be treated as an error condition by the “factory manager”, and a `javax.naming.NoInitialContextException` should be thrown back to the caller.

2. If a specific `InitialContextFactory` implementation is not requested, the “factory manager” should iterate over the list of known implementations of the `javax.naming.spi.InitialContextFactoryBuilder`
interface, and attempt to use each to create the `InitialContextFactory`. If one of the `InitialContextFactoryBuilder` implementations returns a non-null result, this result should be returned to the client. The “factory manager” should return the first non-null result found. The case of a specific factory not being requested is most common in the Java EE use cases, where a client typically uses the no-arguments constructor for `javax.naming.InitialContext`, and the JNDI environment is assumed to be configured by an external container.

3. If an `InitialContextFactory` is not found after iterating over the list of known `InitialContextFactoryBuilder` services, the “factory manager” should try to return a “default” context factory by examining the list of known OSGi services published under the `javax.naming.spi.InitialContextFactory` interface, and return the first service found that implements this interface. The selection of the context factory to be returned should follow the service ranking rules of the OSGi Service Layer.

4. If no implementations exist that can support the given JNDI environment, this should be treated as an error condition by the “factory manager” bundle, and a `javax.naming.NoInitialContextException` should be thrown back to the caller.

The process by which the “factory manager” bundle locates `InitialContextFactory/InitialContextFactoryBuilder` instances has the following implications for JNDI service providers:

1. A service provider wishing to provide a JNDI service that will be specifically requested by the client (using the standard JNDI environment properties) must register an OSGi service under the `javax.naming.spi.InitialContextFactory` interface. This service must also be published under the classname of the factory itself.

2. A service provider wishing to have a more flexible approach to creating JNDI factory instances must register an OSGi service under the `javax.naming.spi.InitialContextFactoryBuilder` interface. This service must examine the parameters passed into the `InitialContextFactoryBuilder.createInitialContextFactory()` method, to determine if this builder can satisfy the given request. If this builder cannot satisfy the given request, the builder method must return null.

3. If any `InitialContextFactory` instances are registered as OSGi services, there is a possibility that one of these instances will be used as a default `InitialContextFactory` as described in the section above.

1.2.1 ObjectFactory integration

The following diagram details how the “factory-manager” bundle can integrate with the JNDI framework to customize the resolution of `javax.naming.Reference` instances.
This diagram shows that the basic integration points with the JNDI framework are quite similar for InitialContextFactoryBuilder and ObjectFactoryBuilder instances. The “factory-manager” bundle registers an implementation of each interface (at most once) with the NamingManager, and these instances are used by the NamingManager at various points to resolve objects. In the case of the InitialContextFactoryBuilder, this delegation occurs at context creation time. In the case of the ObjectFactoryBuilder, this generally occurs when a client attempts a lookup for a name that is bound to a javax.naming.Reference.

The following provides more detail on some of the classes listed in Figure 5-4:

1. **OSGiObjectFactoryBuilder** – an implementation of javax.naming.spi.ObjectFactoryBuilder that the “factory-manager” bundle registers with the NamingManager. JNDI Context implementations are expected to attempt to resolve Reference objects using NamingManager.getObjectInstance(). The proposed implementation will delegate the task of managing references to ObjectFactory to the FactoryManager.

2. **FactoryManager** - This class interacts with OSGi in order to detect any OSGi services that are published that conform to the ObjectFactory interface. This class can utilize the OSGi APIs to detect the published services available, as well as track the registration/de-registration of services during runtime. The presence of this particular class is not necessarily required to implement this bundle. This class is in place to represent the code that will integrate with OSGi in order to detect available JNDI providers. Figure 5-5 below describes how the FactoryManager can register itself with the OSGi service registry in order to locate InitialContextFactoryBuilder and ObjectFactory services.

3. **CustomObjectFactory** – an implementation of javax.naming.ObjectFactory that can be provided by either the “factory-manager” bundle or any bundle deployed in the OSGi framework.

### 1.2.1.1 Behavior of FactoryManager with Object Factories

The FactoryManager class is used in the above diagrams to represent the internal implementation of the factory manager bundle. While the class itself is not required by implementors of this specification, there are some requirements on how this bundle locates and returns JNDI services.

The JNDI framework allows customization at several levels. A service provider typically implements javax.naming.spi.ObjectFactory in order to provide a custom factory for resolving javax.naming.Reference objects. While this is the common case, it is also possible that a service provider may wish to provide an ObjectFactoryBuilder, in order to support more fine-grained control over the creation of context factories. In a non-OSGi environment, providers would likely avoid implementing the builder interface, due to the limitation of only having one ObjectFactoryBuilder registered per VM instance. In the integration with OSGi proposed by this paper, JNDI service providers have the option of registering an ObjectFactoryBuilder that can be queried in order to create an object from a Reference that may not be directly associated with an ObjectFactory classname.

This document proposes that the “factory manager” bundle support both interfaces using the following mechanism:

1. If a javax.naming.Reference passed into the ObjectFactoryBuilder registered by the “factory manager” bundle requires a given factory, the “factory manager” must attempt to locate a service that supports
an interface defined by the classname of the custom factory. The “factory manager” implementation can
determine this factory classname using the Reference.getFactoryClassName(). This service must also
support the javax.naming.spi.ObjectFactory interface. If this service is found, it should be returned to the
client. If a service is not found to be registered under this factory class name, this should be treated as an error
condition by the “factory manager”, and a javax.naming.NamingException should be thrown back to the
caller.

2. If a javax.naming.Reference passed into the ObjectFactoryBuilder registered by the “factory
manager” bundle” does not include a specific ObjectFactory classname to use for reference resolution, the
“factory manager” should iterate over the list of known implementations of the
javax.naming.spi.ObjectFactoryBuilder interface, and attempt to use each to create the
ObjectFactory. If one of the ObjectFactoryBuilder implementations returns a non-null result, this result
should be returned to the JNDI framework. The “factory manager” should return the first non-null result found. If
no implementations exist that can support the given JNDI environment, this should be treated as an error
condition by the “factory manager” bundle, and a javax.naming.NamingException should be thrown back to
the caller.

The process by which the “factory manager” bundle locates ObjectFactory/ObjectFactoryBuilder
instances has the following implications for JNDI service providers:

1. A service provider wishing to provide an ObjectFactory implementation in an OSGi environment must
publish an OSGi service that implements the javax.naming.spi.ObjectFactory interface, as well as the
classname of the custom object factory.

2. A service provider wishing to have a more flexible approach to resolving references must register an OSGi
service under the javax.naming.spi.ObjectFactoryBuilder interface. This service must examine the
parameters passed into the ObjectFactoryBuilder.createInitialContextFactory() method, to
determine if this service can satisfy the given request. If this builder cannot satisfy the request, the builder
method must return null.

The solution proposed in this document would allow for a seamless integration between JNDI clients and JNDI
service providers. JNDI providers could publish their context factories and object factories as OSGi services, and
these implementations will be detected by the “factory-manager” bundle. This approach would also have benefits
for Java EE application server vendors. A standardized bundle of this type in the OSGi framework could
potentially allow for JNDI implementations to be more easily re-used in application servers. This could allow for
application servers to pick up new JNDI implementations without changing code.

1.2.1 Bundle activation

The following diagram details how the “factory-manager” bundle's Activator instance initializes the bundle, and
uses the FactoryManager to obtain the current list of InitialContextFactoryBuilder and ObjectFactoryBuilder
instances, as well as register to be notified when services that implement these interfaces are published or
shutdown.
This diagram demonstrates the activation process of the “factory manager” bundle. The FactoryManager class listed in this diagram is responsible for obtaining JNDI object factories that are published as OSGi services. The OSGi API used to locate the JNDI object factories is not specified, and can be considered an implementation detail.

### 1.2.2 OSGi Service handling in FactoryManager

In the diagrams above, the FactoryManager class is responsible for maintaining a list of available OSGi services that implement the InitialContextFactoryBuilder and ObjectFactory interfaces.

The integration points described above allow the “factory-manager” bundle to make the OSGi services that conform to the InitialContextFactoryBuilder and ObjectFactory interfaces available to any JNDI client code running in the JVM. The dynamic nature of OSGi services opens the possibility that at any time more than one implementation of one of these interfaces may be registered. In the event that multiple services implement the same factory interface, the FactoryManager will observe the same rules for service ranking that OSGi service clients use. These rules are detailed in Chapter 5 (“Service Layer”) of the OSGi Service Platform specification. In the event of multiple implementations of the same interface, the FactoryManager will favor the service with the highest service ranking, then use the lowest service id in the event of a tie. Using this approach will allow JNDI service providers to be used by clients in a predictable fashion, without the possibility of arbitrary factories being returned to the JNDI framework.

### 1.2.3 URL Context Factory Support

Section 3.1.1 of this document explains the basic requirements for URL context factories, and how these factories can be used to locate objects in JNDI using a URL.

The technical solution detailed in Section 5.2 provides an integration point between the JNDI framework and an OSGi bundle that can handle the task of locating and providing builder classes that can create JNDI contexts. Providing the Builder implementations to the javax.naming.spi.NamingManager class is necessary in order to support JNDI running on the OSGi framework, since it allows all context creation requests to be delegated to a single bridge between the JNDI Framework and factory manager bundle.

There is one problem associated with using Builder classes in this way. The support for URL Context Factories in the JNDI framework is provided by the javax.naming.InitialContext class. This class handles the task of...
determining if a URL has been passed to a lookup() request, finding the correct ObjectFactory instance to handle the lookup, and then returning the results to the caller. Once an InitialContextFactoryBuilder implementation has been registered with the NamingManager, all of the default policies of the JNDI framework (including the management of URL context factories) are no longer invoked. At this point, it is the responsibility of the factory manager bundle to ensure that a context implementation returned to the JNDI framework can handle lookup requests for URLs. This support may require wrapping instances of InitialContextFactory as well as the Context interface.

The following diagram details the proposed interaction between the javax.naming.InitialContext instance created by the caller and the factory manager bundle which handles interaction with OSGi in order to obtain JNDI services:

![Sequence Diagram of URL-based lookup using FactoryManager](image)

*Figure 5-6 Sequence Diagram of URL-based lookup using FactoryManager*

The following points are most relevant when considering this diagram:

1. The **Client** class is a JNDI client attempting a lookup using the “util” custom URL scheme. The context being used by the client has already been created and handed back to the client during the call to the javax.naming.InitialContext constructor.

2. The **InitialContext** class acts as a wrapper in many cases for JNDI operations. The internals of this class delegate to the actual context created by the InitialContextFactory.

3. The **default** Context instance queries the FactoryManager to find a factory for the given URL scheme.

4. The **FactoryManager** (described in earlier diagrams) interacts with the OSGi service registry to find a javax.naming.spi.ObjectFactory that can support the requested URL.

5. The **uriContext** instance is used to create a context that supports the “util” scheme, and the lookup request is delegated to that Context instance.
1.2.3.1 Locating URL Context Factory implementations

The default mechanism in the JDK used to find a URL context factory involves checking the value of the "java.naming.factory.url.pkgs" system property. This property contains a list of packages that can be used as a search path for URL context factory implementations. In order to be located by the JNDI framework, implementations must be available on the classpath, and must also include a package prefix in the system property listed above.

This document proposes an additional, more straightforward mechanism for publishing and locating URL context factories:

1. A URL Context factory provider registers an implementation as a service under the `javax.naming.spi.ObjectFactory` interface.

2. This service object is registered with a new service property: "osgi.jndi.urlScheme". This property contains a string representing the URL supported by this ObjectFactory.

3. During requests for URL context factory resolution, the `FactoryManager` can query the service registry for an `ObjectFactory` service that supports the URL scheme passed in during the `Context.lookup()` request.

4. If a matching service is found, the corresponding URL context factory is used to resolve the lookup request.

5. If a matching service is not found, the `FactoryManager` will attempt to invoke `NamingManager.getURLContext()` in order to resolve the URL request.

This approach has the benefit of allowing developers to provide URL context factory implementations as OSGi services. Using the service registry to determine whether a given factory supports a URL scheme provides a simpler mechanism for publishing and locating factory instances. This new proposal removes the package-naming restrictions that are present in the default JDK search process for factory instances.

This approach also provides support the default search method for locating factories. This allows existing URL context factory implementations to be supported without additional changes.

While locating URL context factory implementations in OSGi can ease the restrictions on naming and packaging for a factory implementation, it is important to note that compatibility issues may arise if a given provider is expected to work in both environments. If a service provider wishes to create a URL context factory that can work both in the Java SE environment and the OSGi environment, the provider should continue to follow the packaging and naming guidelines documented by the JDK. This will allow a service provider to publish a URL context factory as an OSGi service while running within a framework, but could also potentially be used in a pure Java environment as well.

1.2.3.1 Returning URL Context Factories to the JNDI Framework

The “factory manager” bundle described in this document is responsible for locating URL context factory implementations, and then providing these factories to JNDI clients through the builder interfaces described earlier. The following support for URL context factories must be supported by the “factory manager”:
1. In order to support the type of lookup sequences listed above, the `javax.naming.spi.InitialContextFactory`, as well as any subsequent contexts created by the factory, must be able to intercept the `lookup()` invocations on the JNDI `Context` by the client.

2. When a client invokes a lookup request using a URL as the lookup string, the JNDI Context that the client interacts with (which is created by the InitialContextFactory that the “factory manager” has returned to the JNDI framework) must detect that a URL has been requested.

3. Once the Context implementation has detected a URL lookup request, this Context is expected to satisfy the lookup request. The process by which the Context implementation interacts with the “factory manager” bundle functionality is not specified, and can be considered an implementation detail.

1.2.3.1 Example of URL Context Factory support

Note: The following section provides details of how an example implementation may be designed to support the requirements for URL Context Factory resolution. This section should not be considered required for implementations of this specification, but is an attempt to describe the most straightforward way of implementing this solution.

In order to support the type of lookup sequences listed above, the `javax.naming.spi.InitialContextFactory`, as well as any subsequent contexts created by the factory, must be wrapped or proxied in order to intercept the `lookup()` invocations on the JNDI `Context` by the client. Building proxy classes for both the Factory and the Context instances is required in this implementation, due to the interaction between the JNDI framework and the `OSGiInitialContextFactoryBuilder` proposed in section 5.2.1.

The following diagram provides detail on the context creation process that will be necessary in order to support URL-based lookups:

![Sequence Diagram of InitialContextFactory Wrapper Creation for URL Support](image-url)
This diagram describes the basic interaction between the JNDI framework and the OSGiInitialContextFactoryBuilder. The "Client" and "InitialContext" objects have been removed from this diagram in order to save space. The NamingManager is attempting to create a new InitialContext. This diagram is similar to Figure 5-3, with the additional calls required to provide wrapper classes for the InitialContextFactory returned by the InitialContextFactoryBuilder, as well as the wrapper class for the javax.naming.Context instance returned by the InitialContextFactory. The "wrapperContext" instance returned by the WrapperContextFactory corresponds to the "default" context instance mentioned in Figure 5-6.

Since the "wrapperContext" will be used to implement support for URL context factories, it is important to note that the underlying context implementation will not be queried if the "wrapperContext" determines that the given lookup string is a URL. This mimics the behavior of javax.naming.InitialContext, since a URL Context Factory can be used with any InitialContextFactory implementation. A URL context factory can be used to extend the lookup behavior of any InitialContextFactory implementation. This also means that when a URL lookup is being handled, either by the default javax.naming.InitialContext implementation or by the method proposed above, the context created by the InitialContextFactory is not used to resolve the lookup. The context implementation itself is ignored during a resolution of a URL lookup.

Note:

The proposal above for URL Context factory support focuses on the most commonly used features, which typically involve passing in a URL to Context.lookup() in order to retrieve an object from the context. The following method:

```java
javax.naming.spi.NamingManager.getURLContext()
```

can be used by JNDI clients in order to create a URL context based on a given scheme. This method does not delegate to either the InitialContextFactoryBuilder or the ObjectFactoryBuilder. This means that the JNDI factory manager in OSGi cannot be registered as a builder for this particular method. This will have to be documented as a limitation of the JNDI integration with OSGi. Since this method is in the service provider package, it is unlikely that application developers will require this functionality on a regular basis.

### 1.2.4 Limitations

Implementing a bundle to handle this integration with JNDI has the following limitations:

1. The “factory-manager” bundle will need to be installed by default in the OSGi framework. Any bundles that require JNDI will need to be started after this bundle has been installed and also initialized.
2. Due to the nature of the integration point with the JNDI framework, any change to this bundle will cause a full restart of the JVM that the framework is running in. Additionally, if the “factory-manager” bundle is stopped, the classes associated with the factory builder plug-in will not be automatically released, since the JNDI framework does not provide a way to reset this builder reference.
3. The “factory-manager bundle” needs to include the “DynamicImport-Package” header in it's manifest file. This is required due to the dynamic nature of JNDI providers. Using this header allows the bundle to respond to any JNDI providers registered on the system, rather than just the bundles that export packages that are imported by the bundle. A component delivered by a JNDI service provider will typically consist of a set of custom factory implementations. The “factory-manager” bundle will not be aware of these types ahead of time. The dynamic import header is required to allow the “factory-manager” bundle to import these types automatically.
4. The “factory-manager” bundle must have the appropriate Java Security permissions to install the InitialContextFactoryBuilder and ObjectFactoryBuilder instances.
1.2.5 Remote JNDI Clients

It should be noted that while most of the examples mentioned in this document refer to JNDI clients accessing services that are local to the OSGi container that the client is running in, the proposed “factory-manager” bundle will also support JNDI clients that wish to connect to remote services. The JNDI service provider is responsible for publishing an InitialContextFactoryBuilder than can create InitialContextFactory instances that provide a remote JNDI context. This implies that the factory can create JNDI contexts that can connect to a remote JNDI server to access bindings. Provided that such an implementation is available, the “factory-manager” will make this context factory available to clients that require this type of connectivity.

1.3 Java EE Requirements in JNDI

1.3.1 Application Namespaces

A Java EE application can access a variety of resources during the course of it's execution. The Java EE Container provides access to resources such as JDBC datasources, JMS Topics and Queues, EJB references, in addition to other types. When a Java EE application creates a new InitialContext, it typically uses the default constructor that takes no arguments. This constructor, with support from the Java EE Container, creates a JNDI context that is suitable for the current application.

The Java EE specification requires that applications have separate namespace boundaries. This means that application “A”s JNDI bindings are treated separately from application “B”s bindings. This allows Java EE applications to define reference and binding names without the complexity of name collisions across all of the applications currently deployed in the container.

OSGi frameworks currently do not support the notion of an application as a logical boundary. This means that Java EE application servers that will build upon the OSGi framework will need to create support for handling this. This document proposes a simple model that could potentially be used to solve this problem.

1.3.1.1 Application Deployment

For any type of namespace boundaries to be observed, a Java EE application server will need to deploy applications in such a way that JNDI contexts can be modeled separately. If, during deployment, an application server deploys a Java EE application as a bundle, the container can register an OSGi service that implements the javax.naming.Context interface. This service would provide a context implementation that represents the bindings in a given application.

Note: As mentioned earlier in the specification, modeling a Java EE application as a bundle is not explicitly required. The scenario described in Section 5.3.1.1 is meant to illustrate how JNDI service providers and Java EE application servers (using the “factory-manager” bundle) can interact in a way that is compliant with standard OSGi.

1.3.1.2 Context discovery

At various points during the runtime, the bundles that comprise the application server container could track the registration/de-registration of these instances, and provide these Context implementations to Java EE applications at runtime. A Servlet Container, for example, could register a ServiceListener for JNDI context-based services.
In order to provide for separation of JNDI namespaces, the Context implementation services should be registered with a set of properties that allow application server components to query for a given application, and perhaps also the version of an application as well. The following properties are proposed:

- `javaee.application.name` - Representing the application name that this Context instance represents
- `javaee.application.version` - Representing the application version that this Context instance represents

Allowing bundles to access a Java EE application's JNDI context provides a standard mechanism for locating an application's Java EE services. This also provides an abstraction layer between the bundles that comprise a Java EE application server and the JNDI contexts available for each deployed application.

### 1.3.1 Environment Naming Context

The Java EE specification requires that application servers provide the “environment naming context” to various components within a Java EE application. The environment naming context differs from a plain JNDI context in that the container must enforce several restrictions imposed by the EE specification. The first restriction is on the namespace of the “java:comp/env” context. This context is considered to be specific to each component deployed in a Java EE container. The namespaces for each component's environment naming context need to logically be considered separate. For example: a servlet in application “A” attempts a lookup for a resource reference (“java:comp/env/test_datasource”) defined in the environment naming context. An EJB in the same application may attempt a lookup for a binding of the same name, but the container must treat these bindings as separate objects. This allows components to define namespaces with less chance of collision. The second most noticeable restriction is that environment naming contexts are always read-only, and the Java EE container is required to guarantee this.

It's not clear yet if any support is required at the OSGi framework layer to support this type of JNDI context. In the future, it may be useful to provide primitive operations that simplify the process of managing these special types of JNDI contexts. JNDI service providers could allow for Context instances to be published as services, with properties indicating that this context has Java EE-specific properties. Application server instances can also manage this context handling internally.

This section of the specification is meant to bring to attention the Java EE requirements for JNDI, which tend to be more restrictive on Java EE application servers than a pure JNDI implementation.

### 1.4 OSGi Service Integration (Still under discussion at the EEG)

#### 1.4.1 JNDI Provider

The OSGi framework will need to provide a default JNDI provider that can be used by clients in order to resolve lookups of OSGi services published to the registry.

#### 1.4.2 Service Access

A JNDI client running inside an OSGi-enabled Java EE Container should be able to access a published OSGi service with the following code example:
InitialContext context = new InitialContext();

ExampleService = (ExampleService) context.lookup("osgi:example_bundle:services/com.example.ExampleService");

In this example, a JNDI client running in an OSGi bundle with a symbolic name of "example_bundle" attempts to lookup an OSGi service with the name "com.example.ExampleService". The client uses the "osgi" URL scheme proposed in this document. One particular area to note is the section of the URL that references the "example_bundle". The JNDI client indicates the bundle symbolic name of the bundle it is running in. This allows the JNDI provider to locate the client bundle's BundleContext, and use that BundleContext instance in order to locate the OSGi service. Using this type of lookup will require clients to specify this bundle symbolic name. Encoding the client's bundle symbolic name in the URL itself has the benefit of not requiring any OSGi-specific code dependencies within the JNDI client itself.

The OSGi framework will be responsible for making a URL context factory implementation available that can handle the "osgi" URL scheme. At initialization time, the framework should create an instance of this URL context factory, and register it under the javax.naming.spi.ObjectFactory interface. The service should also include a the property "osgi.jndi.urlScheme" to indicate that this service supports the "osgi" URL scheme. The packaging details of this "osgi" URL context factory are not specified. This means that this URL context factory could be packaged within the "factory manager" bundle, within the OSGi framework itself, or within another OSGi bundle. Section 5.2.5 of this document provides more detail about support for URL context factories in general.

1.4.2.1 Handling of OSGi Services

Upon a request for a given OSGi service name, the "osgi" URL context factory will use the following method to locate a matching service:

1. The "osgi" URL context factory will attempt to find an OSGi service that is published under the interface specified in the URL. In the example above, the interface to search for would be "com.example.ExampleService".

2. If no services match that interface, the "osgi" URL context factory will attempt to find an OSGi service with the property "osgi.jndi.serviceName" that matches the interface name specified in the URL. For example, a URL with the form "osgi:example_bundle:services/anExampleService" indicates a service name of "anExampleService". This name does not match the interface name of an OSGi service. The "osgi" URL context factory will search for an OSGi service that includes the "osgi.jndi.serviceName" property. Upon finding a service (or services) with this property, the URL context factory will look for a service with an "osgi.jndi.serviceName" that matches the interface name specified in the URL. If a match is found, the URL context factory will return this service to the JNDI framework. If multiple matches are found, the URL context factory will follow the service ranking rules specified in the OSGi Service Layer specification.

In either of the two cases above, the "osgi" URL context factory should ignore any OSGi services that have the following property set:
This OSGi service property allows a service provider to indicate that a given service should not be available via JNDI. By default, the JNDI factory manager will return any matching service it finds that does not have "osgi.jndi.ignoreService" set to "true".

This section includes the following implications for OSGi service providers/implementors:

1. Any OSGi service published to the OSGi registry will be available to JNDI clients at runtime via the service interface.

2. An OSGi service provider that wishes to accommodate mapping an OSGi service to a name that does not match the service interface should publish a given service with the "osgi.jndi.serviceName" property. The "osgi.jndi.serviceName" property should be set to a name that will be used by JNDI clients to locate the service. If the service interface class name is to be used by clients, there is no need to set this property.

3. If an OSGi service provider does not wish to have a service made available to JNDI clients by default, then the "osgi.jndi.ignoreService" property should be set on the service. The "osgi" URL context factory will ignore any services with this property set when requests are made using the "osgi" URL.

The section above proposes two new service properties that are related to JNDI:

1. "osgi.jndi.serviceName" - This property can contain a name that will be used when the "osgi" URL context factory attempts to resolve a lookup of an "osgi"-based URL. This property allows a service to be located with a name other than the classname of the service interface. Usage of this property is optional.

2. "osgi.jndi.ignoreService" - This property can be set by an OSGi service provider that wishes to exclude a given service from the search order of the "osgi" URL context factory. Setting this property to "true" on a service will force the "osgi" URL context factory to ignore this service when resolving lookups.

The URL context factory implementation for the "osgi" URL scheme will interact with the OSGi Service API in order to locate a service, as well as receive updates on that service's status.

1.4.2.1 Example of Service Handling using Proxies

Note: This section is meant to demonstrate a possible approach towards supporting OSGi service access via JNDI. This particular approach is not required to be used by implementors of the specification.

Before returning a service implementation to the caller, the "osgi" URL Context Factory will generate a wrapper proxy for the service in order to manage the interaction with the underlying OSGi framework.
This proxy could be implemented in several ways, such as:

- Dynamic Proxies used as the wrapper
- A binary library used to dynamically create a class that wraps the Service Reference, such as BCEL or ASM

Maintaining this level of indirection between the service and the JNDI client will allow for the following:

- **Limited Lifecycle Management**: The JNDI Context interface does not provide a mechanism for releasing a resource. Using a proxy approach in this case allows the JNDI implementation to determine when the service is garbage-collected. This provides at least one way to release the resource when the client no longer uses the service. *Note: This approach relies on garbage collection, which is not always guaranteed to occur. This approach is meant as a compromise between the JNDI programming model and the OSGi services model.*

- **General Service Management**: The service reference returned to the caller may, at any time, be deregistered from the OSGi service registry. Having a proxy wrapper allows the JNDI implementation to use the OSGi Service API to detect these scenarios, and throw the correct exceptions. This proxy approach may also allow for the underlying service to be stopped and restarted without the JNDI client having to refresh its reference.

### 1.4.1 “osgi” URL Support

The “osgi” URL syntax to be used for resolving OSGi service references via JNDI is as follows:

```
osgi:bundle_symbolic_name:services:<interface>/<filter>
```

- The scheme of this URL is “osgi”.
- “bundle_symbolic_name” indicates the symbolic name of the bundle that the JNDI client is running in
- “services” indicates that this namespace only includes published OSGi services
- `<interface>` contains the name of an OSGi service that a JNDI client wishes to locate. This name may be the actual name of the service interface type, or may be any arbitrary string.
- `<filter>` contains any service-query filter information. This filter should follow the same syntax that is already present in the OSGi service model.
1 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.

1.1 URL Stream Handler Service Analysis

Since the URL Handler Service in OSGi already handles integration support for different URL schemes in an OSGi container, some investigation was done to determine if there was any commonality between the URL Stream Handler integration and the work required by this RFC in order to support URL Context Factories. It was ultimately determined that the APIs and semantics for each framework are not similar enough to foster any re-use that would be helpful in implementing support for URL Context factories. The following sections document the results of this investigation.

1.1.1 Similarities with URL Handler Service Specification

Chapter 11 of the OSGi Specification deals with the integration of Java’s URL handler facilities with the OSGi framework. This chapter details some of the difficulties in dealing with some of the integration points in Java for URL handling and content type handling. The key problem lies in the fact that many of the object factory frameworks in Java do allow for a plug-in replacement, but the plug-in typically can only be set once per JVM invocation. This can cause problems with classloading, since classes that are no longer used may not be garbage-collected, since the Java URL handling framework caches references to various factories.

The JNDI/OSGi integration encounters many of the same issues when registering InitialContextFactoryBuilder and ObjectFactoryBuilder instances with the NamingManager. This specification uses a similar approach of registering one factory that can be used to delegate to any number of factories.

1.1.2 Differences with URL Handler Service Specification

1.1.2.1 Programming Interface

One immediate difference between the JNDI’s support for URL context factories and the JDK’s support for URL Stream handlers is that each framework provides a separate API for URL support.

Adding support for a new URL handler scheme in the JDK requires subclassing java.net.URLStreamHandler as well as java.net.URLConnection, while adding support for a new URL context factory in JNDI requires that a developer provide an implementation of javax.naming.spi.ObjectFactory. The URL handler API focuses on the creation of a URLConnection, while the ObjectFactory API only requires the creation of an object or JNDI context based on the contents of a URL. A factory implemented for URL stream handler support will not be interchangeable with a URL context factory. In the same way, a URL context factory will not be interchangeable with a URL stream handler.
1.1.2.2 **URL Handler/ObjectFactory locations**

While both framework options define a package structure and class-naming scheme for URL support, the requirements are different in each framework.

The javadoc for `java.net.URL` defines the packaging requirements for a `URLStreamHandler` implementation:

http://java.sun.com/j2se/1.5.0/docs/api/java/net/URL.html

The javadoc for `javax.naming.spi.NamingManager.getURLContext()` defines the packaging requirements for a URL context factory implementation:

http://java.sun.com/j2se/1.5.0/docs/api/javax/naming/spi/NamingManager.html#getURLContext(java.lang.String,%20java.util.Hashtable)

1.1.2.3 **Service wrapping**

The JNDI/OSGi integration differs from the URL Handler Service Specification in that the interfaces published by JNDI service providers do not need to be wrapped in a service facade. The URL Handler framework must wrap the URL handler interfaces due to access restrictions. The JNDI integration does not require this extra service layer.

---

### 2 Security Considerations

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

Obtaining a reference to a JNDI `Context` or an `InitialContextFactory` should be considered a privileged operation and should be guarded by permissions.

The “factory-manager” bundle must be given additional permissions in order to locate and utilize any JNDI providers published as OSGi services. This bundle must have the “GET” permission for every implementation of the JNDI services published in OSGi service registry (`javax.naming.spi.InitialContextFactoryBuilder`, `javax.naming.spi.ObjectFactoryBuilder`, `javax.naming.Context`). This bundle will exist at the system layer, and as such should have more permissions to access services than a standard bundle.
3 Document Support

3.1 References


3.1 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>John Wells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Oracle</td>
</tr>
<tr>
<td>Address</td>
<td>150 Allen Road, Liberty Corner NJ 07938</td>
</tr>
<tr>
<td>Voice</td>
<td>(908) 580-3127</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:john.wells@oracle.com">john.wells@oracle.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Robert W. Nettleton</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Oracle</td>
</tr>
<tr>
<td>Address</td>
<td>330 Fellowship Road Suite 100, Mt. Laurel NJ, 08054</td>
</tr>
<tr>
<td>Voice</td>
<td>(856) 359-2927</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:bob.nettleton@oracle.com">bob.nettleton@oracle.com</a></td>
</tr>
</tbody>
</table>

3.2 Acronyms and Abbreviations

3.3 End of Document
The OSGi Alliance and its members specify, create, advance, and promote wide industry adoption of an open delivery and management platform for application services in home, commercial buildings, automotive and industrial environments. The OSGi Alliance serves as the focal point for a collaborative ecosystem of service providers, developers, manufacturers, and consumers. The OSGi specifications define a standardized, component oriented, computing environment for networked services. OSGi technology is currently being delivered in products and services shipping from several Fortune 100 companies. The OSGi Alliance’s horizontal software integration platform is ideal for both vertical and cross-industry business models within home, vehicle, mobile and industrial environments. As an independent non-profit corporation, the OSGi Alliance also provides for the fair and uniform creation and distribution of relevant intellectual property – including specifications, reference implementations, and test suites – to all its members.

HOW TO REACH US:

OSGi Alliance
Bishop Ranch 6
2400 Camino Ramon, Suite 375
San Ramon, CA 94583 USA

Phone: +1.925.275.6625
E-mail: marketinginfo@osgi.org
Web: http://www.osgi.org

OSGi is a trademark of the OSGi Alliance in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

All other marks are trademarks of their respective companies.