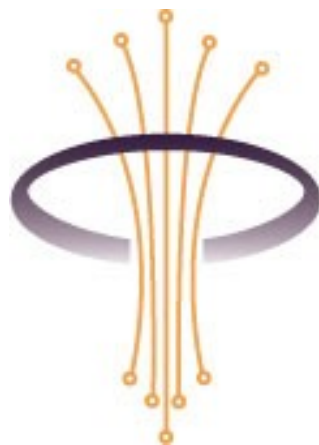




OSGi Service Platform Release 4

Version 4.2 - Early Draft

Revision 1.0
5 August 2008



OSGi[™]
Alliance

DISTRIBUTION AND FEEDBACK LICENSE

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

DATE OF DISTRIBUTION: 6 August 2008

© 2008 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 the Early Draft 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 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 comments can be mailed to speccomments@mail.osgi.org.

BJ Hargrave
Chief Technical Officer
OSGi Alliance



Core Design Documents

OSGi Service Platform Release 4

Version 4.2 - Early Draft

Revision 1.0
5 August 2008



RFC 120 - Security Enhancements

Draft

40 Pages

Abstract

This RFC proposes the ability to deny access to resources. The current system only allows the granting of privileges. Adding the ability to deny privileges leads to simpler administration of security in many use cases and hence a more secure system.

Copyright © 2008.

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	3
0.3 Revision History	3
1 Introduction	4
2 Application Domain	5
2.1 Terminology + Abbreviations	5
3 Problem Description	5
4 Requirements	6
5 Technical Solution	6
5.1 Ordered Table with Decision Column Discussion	6
5.2 Conditional Permission Table Requirements	8
5.3 Negative Condition Requirements	10
5.4 Atomic API Discussion	10
5.5 Javadoc	11
5.5.1 org.osgi.service.condpermadmin Class BundleLocationCondition	11
5.5.2 org.osgi.service.condpermadmin Class BundleSignerCondition	12
5.5.3 org.osgi.service.condpermadmin Interface Condition	13
5.5.4 org.osgi.service.condpermadmin Interface ConditionalPermissionAdmin	16
5.5.5 org.osgi.service.condpermadmin Interface ConditionalPermissionInfo	19
5.5.6 org.osgi.service.condpermadmin Interface ConditionalPermissionInfoBase	20
5.5.7 org.osgi.service.condpermadmin Interface ConditionalPermissionsUpdate	22
5.5.8 org.osgi.service.condpermadmin Class ConditionInfo	24
5.6 Open Issues	27
5.7 Closed Issues	27
5.7.1 Default Decision	27
5.7.2 An Alternate Update model	28
5.7.3 Utility methods	29
6 Considered Alternatives	29
6.1 com.bea.sandbox.security.permission Class DeniablePermission	29
6.1.1 DeniablePermission	31
6.1.2 DeniablePermission	31

6.1.3 implies31

6.1.4 hashCode.....32

6.1.5 equals.....32

6.1.6 toString.....32

6.2 Deny Permission Column32

6.2.1 Add Deny Column to Permission Table Discussion32

6.2.2 NOT Condition Discussion.....34

6.2.3 NotCondition Requirements.....35

6.2.4 NotCondition Javadoc.....36

org.osgi.service.condpermadmin Class NotCondition.....36

getCondition.....36

6.2.5 Other API modifications37

6.2.6 org.osgi.service.condpermadmin.ConditionInfo37

ConditionInfo37

isNot37

6.2.7 org.osgi.service.condpermadmin.ConditionPermissionInfo.....37

getDenyPermissionInfos38

6.2.8 org.osgi.service.condpermadmin.ConditionalPermissionAdmin38

addConditionalPermissionInfo38

setConditionalPermissionInfo38

6.2.9 Issues.....39

6.2.10 Friends API39

7 Security Considerations40

8 Document Support40

8.1 References.....40

8.2 Author’s Address40

8.3 Acronyms and Abbreviations.....40

8.4 End of Document.....40

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.

Revision	Date	Comments
Initial	Aug 8, 2007	Initial revision. Covers only a few enhancements as well as a discussion of a mechanism that has been considered but rejected John Wells, BEA Systems, Inc., jwells@bea.com

Revision	Date	Comments
0.2	Dec 14, 2007	Clean up document after initial review. Add support for Boolean condition expressions. Include more use case examples.
0.3	Jan. 10, 2008	Yet another change in direction
0.4	Feb. 7, 2008	Remove ABSTAIN and create an atomic update API
0.5	Feb. 15, 2008	Change to a different style of update API
0.6	Mar. 10, 2008	Cleanup to the API based on review comments
0.7	May 7, 2008	More API cleanup
0.8	Jun 25, 2008	Add better algorithm description, fix how postponed conditions should work and fix the example to make more sense
0.9	Jul 23, 2008	Algorithm for postponed conditions
0.91	Aug 05, 2008	Changes to simplify postponed conditions algorithm + updated javadoc Thomas Watson, IBM, tjwatson@us.ibm.com

1 Introduction

One reasonable property of many “enterprise” servers is that they host user code along with vendor supplied “system” code within the same Java VM. In order to do this there needs to be some reasonable way to allow system code to separate itself from the user code so that the system code can safely offer services, packages and other resources to which it does not wish user code to have access.

This RFC is a response to RFP-78 ([3]). It extends the existing security tuple table, while maintaining backward compatibility with all existing API and security file formats.

2 Application Domain

From [3]:

The application domain is OSGi server applications where third party bundles may be deployed alongside trusted bundles from the application server provider. The ability to split the set of bundles into categories like these is already well defined in the Conditional Permission Admin Service.

2.1 Terminology + Abbreviations

- **Application Server:** In this paper an application server refers to a running OSGi system that has trusted system bundles written by the application server vendor running alongside third-party user bundles that are provided by the end customer (i.e., not the application server vendor). It does not imply a JEE, .NET or database server, but simply a server whose job it is to host other services and provide them a consistent set of system services upon which they can rely.
- **Conditional Permission Service:** This is the service defined in Chapter 9 of the OSGi Service Platform Core Specification (R4). It defines conditions, permissions and how security is applied in the R4 OSGi platform.

3 Problem Description

From [3]:

The existing permission classes in OSGi do not have the flexibility needed to handle certain use cases. This includes but is not limited to

- `org.osgi.framework.PackagePermission`
- `org.osgi.framework.ServicePermission`
- `org.osgi.service.event.TopicPermission`
- `org.osgi.service.wireadmin.WirePermission`
- `org.osgi.service.cm.ConfigurationPermission`

Providing these capabilities on the OSGi platform will facilitate the adoption of OSGi into enterprise application server environments.

4 Requirements

From [3]:

1. The solution **MUST** enable restricting certain bundles from offering a service, without having to list all of the services that bundle might offer.
2. The solution **MUST** enable restricting certain bundles from getting a reference to a service, without having to list all of the other services that bundle might want access to.
3. The solution **MUST** enable restricting certain bundles from importing or exporting a package, without having to list all of the packages a bundle might import or export.
4. The solution **MAY** allow for a general Permission denial model
5. The solution **MUST NOT** alter the bundle programming model
6. The solution **MAY** boost performance by allowing for Permission implies decisions to be cached
7. Any RFC written **MUST** support all existing OSGi execution environments

5 Technical Solution

5.1 Ordered Table with Decision Column Discussion

In this solution we change the security table in two ways. Firstly, we make it an ordered table. Secondly, we add another column to the table which determines if the decision of the row is to “allow” access to the resource or “deny” access to the resource.

A few example use cases will make this clear.

In the first use case ACME, a provider of system software, would like to allow all other bundles access to the com.acme package family but restrict them from accessing the “com.acme.secret” and “com.acme.sauce” packages.

Name	Conditions	Permissions	Decision
R1	NOT Signed by ACME	Package(“com.acme.secret”) Package(“com.acme.sauce”)	DENY
R2	Signed by ACME	Package(“com.acme.*”)	ALLOW
R3	<Empty>	Package(“com.acme.*”)	ALLOW

Suppose ACME is attempting to access package “com.acme.secret”. Row R1 will not be evaluated since the condition does not match. Row R2 will be evaluated, and since Package(“com.acme.*”) implies “com.acme.secret” this check will succeed.

Now suppose Iona is attempting to access package “com.acme.secret”. Row R1 would be evaluated and since Package(“com.acme.secret”) implies “com.acme.secret” this check would return DENY and would fail as expected.

If instead Iona is attempting to access package “com.acme.service” then Row R1 would be evaluated. However, since neither Package(“com.acme.secret”) nor Package(“com.acme.sauce”) implies “com.acme.service” the row will not be evaluated (in effect the decision of this row is to ABSTAIN). Row R2 will be skipped (since the condition does not match) and row R3 will be evaluated. Since Package(“com.acme.*”) implies “com.acme.service” the decision of ALLOW will be taken and the permission check will succeed.

Note that the next table is equivalent to the table above:

Name	Conditions	Permissions	Decision
R1	NOT Signed by ACME	Package(“com.acme.secret”) Package(“com.acme.sauce”)	DENY
R2	<Empty>	Package(“com.acme.*”)	ALLOW

One use case (which is not in the original set of use cases) that is handled cleanly by this solution is the friends use case. In this use case we want to express the following scenario:

1. Pepsi wants to deny permissions to com.pepsi.* for everyone but Pepsi
2. Pepsi wants to allow Coke to have permission to com.pepsi.friends.*
3. All people should have access to all other packages

The following table enables the above use case:

Name	Conditions	Permissions	Decision
------	------------	-------------	----------

Name	Conditions	Permissions	Decision
R1	Signed by Coke	Package("com.pepsi.friends.*")	ALLOW
R2	Not Signed by Pepsi	Package("com.pepsi.*")	DENY
R3	<Empty>	Package("**")	ALLOW

The reason that the above use case works is that the table is now ordered, and will be evaluated in row order.

If Coke attempts to access the "com.pepsi.friends.foo" package then row R1 will be evaluated first and since Package("com.pepsi.friends.*") implies package "com.pepsi.friends.foo" the ALLOW decision is taken and the check will succeed.

If Coke attempts to access the "com.pepsi.secret" package then row R1 will be evaluated first and since Package("com.pepsi.friends.*") does not imply package "com.pepsi.secret" the row is not evaluated (in essence row R1 abstains from the decision). Row R2 also applies to Coke and since Package("com.pepsi.*") implies package "com.pepsi.secret" the decision of DENY is found and hence this check would fail (properly).

If I am Pepsi on the other hand, if I attempt to access package "com.pepsi.friends" or package "com.pepsi.secret" then neither rows R1 or R2 apply, and only row R3 will be considered. Since Package("**") implies both the packages listed above the decision of ALLOW is found and hence this check would succeed.

Also, RC Cola, who is *not* a friend of Pepsi, does not have access to either "com.pepsi.friends.*" or "com.pepsi.*" but does have access to all the other packages in the system.

5.2 Conditional Permission Table Requirements

A fourth column shall be added to the conceptual Conditional Permission table. The cells in this column shall take a single java.lang.String object. The allowable values for this string shall be "allow" or "deny". The allowable decision strings shall be added to the org.osgi.service.condperadmin.ConditionalPermissionInfoBase class.

Previously the order of the rows in the conceptual Conditional Permission table were not significant. The rows of the conceptual Conditional Permission table shall now be ordered by the index of the row. In other words, all rows shall be evaluated in index order (lowest to highest). This includes rows with postponed conditions.

A permission check starts when the Security Manager checkPermission method is called with permission P as argument. This Security Manager must be implemented by the Framework and is therefore called the Framework Security Manager; it must be fully integrated with the Conditional Permission Admin service.

The Framework Security Manager must get the Access Control Context in effect. It must call the AccessController getContext() method to get the default context if it is not passed a specific context.

The AccessControlContext checkPermission method must then be called, which causes the call stack to be traversed. At each stack level the Bundle Protection Domain of the calling class is evaluated for the permission P using the ProtectionDomain implies method. This complete evaluation must take place on the same thread.

P must be implied by the local permissions of the Bundle Protection Domain. If this is not the case, the check must end with a failure.

The Bundle Protection Domain must now decide which rows in its instantiated conditional permission table are applicable and imply P.

It must therefore execute the following instructions or reach the same result in an alternative way:

- For each row R in the domain's instantiated conditional permission table:
 - If R has immediate conditions, evaluate all these immediate Condition objects. If any of these objects is not satisfied, continue with next row.
 - If R's permissions do not imply P, continue with the next row.
 - If R contains postponed Condition objects then add R to the end of the postpone list for the domain's instantiated conditional permission table (which will postpone the evaluation of R) and continue with the next row.
 - Otherwise:
 - If there are no rows have been postponed for the domain then return true if R's decision is Grant; otherwise return false if R's decision is Deny.
 - Otherwise the postponed list for the domain's instantiated conditional permission table shall be inspected from the END to the START removing all postponed rows that have the same decision as R's decision until a postponed row is encountered with the opposite decision
 - If all postponed rows are removed then return true if R's decision is Grant; otherwise return false if R's decision is Deny.
 - Otherwise the remaining postponed rows along with R's immediate decision must be postponed and true must be returned
- After all rows have been processed and no immediate decision has been found
 - If there were any postponements, then return true. Otherwise, return false.

After the Framework Security Manager has called the checkPermission method of the Access Control Context, it must decide to fail or handle the postponed rows for each domain. If this method returns false, then the Framework Security Manager's checkPermission method fails.

If it returns true, there could still be a list of postponed rows for each Bundle Protection Domain's instantiated conditional permission table. Each of these postponed rows already imply permission P, otherwise they must not have been placed on the postponed list for the domain's instantiated conditional permission table. However, their Condition objects still need to be satisfied before the decision of the row can apply to the security check.

Frameworks are free to implement an algorithm that finds optimal ways to permute the postponed rows from the different domains involved in the permission check. The end result must return the same answer as the following algorithm for processing postponed conditions.

- For each domain D that has and order list of postponed rows
 - For each postponed row R
 - For each postponed condition C
 - Call Condition.isSatisfied(Condition[], Dictionary) with the single condition instance C. Each call to Condition.isSatisfied(Condition[], Dictionary) for a single condition type must use the same Dictionary instance during a single permission check.
 - If Condition.isSatisfied(Condition[], Dictionary) returns false then continue to the next postponed row; Otherwise continue to the next condition.
 - If the decision is Grant then continue to the next domain
 - If the decision is Deny then throw a SecurityException

- If there was an immediate decision recorded along with the postponed rows of domain D then
 - If the immediate decision is Grant continue to the next domain
 - If the decision is Deny then throw a SecurityException
 - Otherwise no rows apply to the permission check for this domain; a SecurityException must be thrown.
- If every domain is processed without throwing a SecurityException then the permission is granted.

The algorithm described above is descriptive and other implementations of how the postponed lists work may be used, as long as the end goal of having the security table behave as if it is ordered (even if the conditions are not actually evaluated in the given order) is achieved.

5.3 Negative Condition Requirements

The `org.osgi.service.condpermadmin.BundleLocationCondition` and `org.osgi.service.condpermadmin.BundleSignerCondition` shall have an optional second string added to their initialization arguments. If this string is equal to "!" then the Condition shall return the logical NOT of the result of the match of the first string (the DN in the case of the `BundleSignerCondition` and the bundle location in the case of the `BundleLocationCondition`).

5.4 Atomic API Discussion

In order to support an ordered permission table it is necessary to support an atomic style of updating that table. In particular multiple entries may need to be added or removed from the table in a single operation in order to avoid either granting or denying too many rights. However, the original API was not designed for atomic update. In particular, the "delete" method on the `ConditionalPermissionInfo` objects as the means to remove items from the table does not lend itself to atomicity.

In order to support atomic updates to the table a new object called the `ConditionalPermissionAdminUpdate` (or just "Update" for short) has been created. The Update is originally created with a copy of the existing Permission Table. There may be any number of Update objects in the system based on the running Permission Table. However an Update will only successfully commit if the running permission table has not changed.

The `ConditionalPermissionInfo` interface has been split into two interfaces, the `ConditionalPermissionInfoBase` interface and the existing `ConditionalPermissionInfo` interface. All of the methods previously in the `ConditionalPermissionInfo` interface have been moved to the `ConditionalPermissionInfoBase` interface with the exception of the delete method, which has remained in the `ConditionalPermissionInfo` interface. The `ConditionalPermissionInfo` interface extends the `ConditionalPermissionInfoBase` interface. In this way both source and binary compatibility is achieved, while still allowing an atomic coding paradigm. (Note that source compatibility does **not** extend to the use of reflection, which may now get different results).

The Update's List can be modified by the code without affecting the currently running permission table. At the time the commit method is called on the Update's list will become the new permission table as long as no other update has been committed on the Permission Table since the Update was created. There is no requirement that the commit method be called on an Update object.

While the Update mechanism helps satisfy the requirement for atomic updates to the permission table, there are some idiosyncrasies associated with the approach. For example, since there is no lock it is not possible to write code using this API that can be completely isolated from the possibility of losing the Update race. Therefore to be

completely correct all code using this new API must handle the case where the Update race is lost. This may or may not be difficult to achieve in general code.

The exact mechanism and behavior for the atomic updates of the permission table can be found in the javadoc section of this document.

5.5 Javadoc

Several classes need to be modified to support the specified requirements. The javadoc should be considered a binding part of this specification. Note that only API that has been modified or added is included in this document. All API not in this document shall continue to behave as per prior art.

Package	Class	Use Tree	Deprecated	Index	Help			
PREV CLASS	NEXT CLASS					FRAMES	NO FRAMES	All Classes
SUMMARY: NESTED FIELD CONSTR METHOD						DETAIL: FIELD CONSTR METHOD		

5.5.1 org.osgi.service.condpermadmin Class BundleLocationCondition

```
java.lang.Object
└─ org.osgi.service.condpermadmin.BundleLocationCondition
```

```
public class BundleLocationCondition
extends java.lang.Object
```

Condition to test if the location of a bundle matches or does not match a pattern. Since the bundle's location cannot be changed, this condition is immutable.

Pattern matching is done according to the filter string matching rules.

Version:
\$Revision: 5185 \$

Method Summary

<code>static</code>	<code>Condition getCondition(Bundle bundle, ConditionInfo info)</code>
	Constructs a condition that tries to match the passed Bundle's location to the location pattern.

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Method Detail

5.5.1.1 getCondition

```
public static Condition getCondition(Bundle bundle,
                                     ConditionInfo info)
```

Constructs a condition that tries to match the passed Bundle's location to the location pattern.

Parameters:

`bundle` - The Bundle being evaluated.

`info` - The ConditionInfo from which to construct the condition. The ConditionInfo must specify one or two arguments. The first argument of the ConditionInfo specifies the location pattern against which to match the bundle location. Matching is done according to the filter string matching rules. Any "*" characters in the first argument are used as wildcards when matching bundle locations unless they are escaped with a '\' character. The Condition is satisfied if the bundle location matches the pattern. The second argument of the ConditionInfo is optional. If a second argument is present and equal to "!", then the satisfaction of the Condition is negated. That is, the Condition is satisfied if the bundle location does NOT match the pattern. If the second argument is present but does not equal "!", then the second argument is ignored.

Returns:

Condition object for the requested condition.

[Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.5.2 org.osgi.service.condpermadmin Class BundleSignerCondition

java.lang.Object

 org.osgi.service.condpermadmin.BundleSignerCondition

```
public class BundleSignerCondition
extends java.lang.Object
```

Condition to test if the signer of a bundle matches or does not match a pattern. Since the bundle's signer can only change when the bundle is updated, this condition is immutable.

The condition expressed using a single String that specifies a Distinguished Name (DN) chain to match bundle signers against. DN's are encoded using IETF RFC 2253. Usually signers use certificates that are issued by certificate authorities, which also have a corresponding DN and certificate. The certificate authorities can form a chain of trust where the last DN and certificate is known by the framework. The signer of a bundle is expressed as signers DN followed by the DN of its issuer followed by the DN of the next issuer until the DN of the root certificate authority. Each DN is separated by a semicolon.

A bundle can satisfy this condition if one of its signers has a DN chain that matches the DN chain used to construct this condition. Wildcards (^*) can be used to allow greater flexibility in specifying the DN chains. Wildcards can be used in place of DN's, RDN's, or the value in an RDN. If a wildcard is used for a value of an RDN, the value must be exactly "*" and will match any value for the corresponding type in

that RDN. If a wildcard is used for a RDN, it must be the first RDN and will match any number of RDNs (including zero RDNs).

Version:

\$Revision: 5185 \$

Method Summary

<pre>static Condition getCondition(Bundle bundle,</pre>	<pre> ConditionInfo info) Constructs a Condition that tries to match the passed Bundle's location to the location pattern.</pre>
---	---

Methods inherited from class java.lang.Object

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

Method Detail

5.5.2.1 getCondition

```
public static Condition getCondition(Bundle bundle,
                                     ConditionInfo info)
```

Constructs a Condition that tries to match the passed Bundle's location to the location pattern.

Parameters:

bundle - The Bundle being evaluated.

info - The ConditionInfo from which to construct the condition. The ConditionInfo must specify one or two arguments. The first argument of the ConditionInfo specifies the chain of distinguished names pattern to match against the signer of the bundle. The Condition is satisfied if the signer of the bundle matches the pattern. The second argument of the ConditionInfo is optional. If a second argument is present and equal to "!", then the satisfaction of the Condition is negated. That is, the Condition is satisfied if the signer of the bundle does NOT match the pattern. If the second argument is present but does not equal "!", then the second argument is ignored.

Returns:

A Condition which checks the signers of the specified bundle.

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.5.3 org.osgi.service.condpermadmin Interface Condition

```
public interface Condition
```

The interface implemented by a Condition. Conditions are bound to Permissions using Conditional Permission Info. The Permissions of a ConditionalPermission Info can only be used if the associated Conditions are satisfied.

Version:

\$Revision: 5184 \$

Field Summary

static Condition FALSE	A Condition object that will always evaluate to false and that is never postponed.
static Condition TRUE	A Condition object that will always evaluate to true and that is never postponed.

Method Summary

boolean isMutable()	Returns whether the Condition is mutable.
boolean isPostponed()	Returns whether the evaluation must be postponed until the end of the permission check.
boolean isSatisfied()	Returns whether the Condition is satisfied.
boolean isSatisfied (Condition [] conditions, java.util.Dictionary context)	Returns whether a the set of Conditions are satisfied.

Field Detail

5.5.3.1 TRUE

static final [Condition](#) **TRUE**
A Condition object that will always evaluate to true and that is never postponed.

5.5.3.2 FALSE

static final [Condition](#) **FALSE**
A Condition object that will always evaluate to false and that is never postponed.

Method Detail

5.5.3.3 isPostponed

boolean [isPostponed\(\)](#)

Returns whether the evaluation must be postponed until the end of the permission check. This method returns `true` if the evaluation of the Condition must be postponed until the end of the permission check. If this method returns `false`, this Condition must be able to directly answer the [isSatisfied\(\)](#) method. In other words, `isSatisfied()` will return very quickly since no external sources, such as for example users, need to be consulted.

Returns:

`true` to indicate the evaluation must be postponed. Otherwise, `false` if the evaluation can be immediately performed.

5.5.3.4 isSatisfied

boolean `isSatisfied()`

Returns whether the Condition is satisfied.

Returns:

`true` to indicate the Conditions is satisfied. Otherwise, `false` if the Condition is not satisfied.

5.5.3.5 isMutable

boolean `isMutable()`

Returns whether the Condition is mutable.

Returns:

`true` to indicate the value returned by [isSatisfied\(\)](#) can change. Otherwise, `false` if the value returned by [isSatisfied\(\)](#) will not change.

5.5.3.6 isSatisfied

boolean `isSatisfied`([Condition](#)[] conditions,
 java.util.Dictionary context)

Returns whether a the set of Conditions are satisfied. Although this method is not static, it must be implemented as if it were static. All of the passed Conditions will be of the same type and will correspond to the class type of the object on which this method is invoked.

Parameters:

`conditions` - The array of Conditions.

`context` - A Dictionary object that implementors can use to track state. If this method is invoked multiple times in the same permission evaluation, the same Dictionary will be passed multiple times. The SecurityManager treats this Dictionary as an opaque object and simply creates an empty dictionary and passes it to subsequent invocations if multiple invocations are needed.

Returns:

`true` if all the Conditions are satisfied. Otherwise, `false` if one of the Conditions is not satisfied.

Package **Class** **Use Tree** **Deprecated** **Index** **Help**

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Package **Class** **Use Tree** **Deprecated** **Index** **Help**

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.5.4 org.osgi.service.condpermadmin Interface ConditionalPermissionAdmin

public interface **ConditionalPermissionAdmin**

Framework service to administer Conditional Permissions. Conditional Permissions can be added to, retrieved from, and removed from the framework. Conditional Permissions are conceptually managed in an ordered table called the Conditional Permission Table.

Version:

\$Revision: 5188 \$

Method Summary

ConditionalPermissionInfo	addConditionalPermissionInfo (ConditionInfo [] conds, PermissionInfo [] perms) Deprecated. Since 1.1. Use ConditionalPermissionsUpdate instead.
ConditionalPermissionInfoBase	createConditionalPermissionInfoBase (java.lang.String name, ConditionInfo [] conditions, PermissionInfo [] permissions, java.lang.String decision) Creates a ConditionalPermissionInfoBase with the specified fields.
ConditionalPermissionsUpdate	createConditionalPermissionsUpdate () Creates an update for the Conditional Permission Table.
Java.security.AccessControlContext	getAccessControlContext (java.lang.String[] signers) Returns the Access Control Context that corresponds to the specified signers.
ConditionalPermissionInfo	getConditionalPermissionInfo (java.lang.String name) Return the Conditional Permission Info with the specified name.
java.util.Enumeration	getConditionalPermissionInfos () Returns the Conditional Permission Infos from the Conditional Permission
ConditionalPermissionInfo	setConditionalPermissionInfo (java.lang.String name, ConditionInfo [] conds, PermissionInfo [] perms) Deprecated. Since 1.1. Use ConditionalPermissionsUpdate instead.

Method Detail

5.5.4.1 addConditionalPermissionInfo

[ConditionalPermissionInfo](#) [addConditionalPermissionInfo](#)([ConditionInfo](#)[] conds, [PermissionInfo](#)[] perms)

Deprecated. Since 1.1. Use [ConditionalPermissionsUpdate](#) instead.

Create a new Conditional Permission Info in the Conditional Permission Table.

The Conditional Permission Info will be given a unique, never reused name. This entry will be added at the beginning of the Conditional Permission Table with a grant decision of [ALLOW](#).

Since this method changes the Conditional Permission Table any [ConditionalPermissionsUpdates](#) that were created prior to calling this method can no longer be committed.

Parameters:

`conds` - The Conditions that need to be satisfied to enable the corresponding Permissions.

`perms` - The Permissions that are enabled when the corresponding Conditions are satisfied.

Returns:

The `ConditionalPermissionInfo` for the specified Conditions and Permissions.

Throws:

`java.lang.SecurityException` - If the caller does not have `AllPermission`.

5.5.4.2 *setConditionalPermissionInfo*

[ConditionalPermissionInfo](#) `setConditionalPermissionInfo`(`java.lang.String` name, `ConditionInfo[]` conds, `PermissionInfo[]` perms)

Deprecated. *Since 1.1. Use [ConditionalPermissionsUpdate](#) instead.*

Set or create a Conditional Permission Info with a specified name in the Conditional Permission Table.

If the specified name is `null`, a new Conditional Permission Info must be created and will be given a unique, never reused name. If there is currently no Conditional Permission Info with the specified name, a new Conditional Permission Info must be created with the specified name. Otherwise, the Conditional Permission Info with the specified name must be updated with the specified Conditions and Permissions. If a new entry was created in the Conditional Permission Table it will be added at the beginning of the table with a grant decision of [ALLOW](#).

Since this method changes the underlying permission table any [ConditionalPermissionsUpdates](#) that were created prior to calling this method can no longer be committed.

Parameters:

`name` - The name of the Conditional Permission Info, or `null`.

`conds` - The Conditions that need to be satisfied to enable the corresponding Permissions.

`perms` - The Permissions that are enabled when the corresponding Conditions are satisfied.

Returns:

The `ConditionalPermissionInfo` that for the specified name, Conditions and Permissions.

Throws:

`java.lang.SecurityException` - If the caller does not have `AllPermission`.

5.5.4.3 *getConditionalPermissionInfos*

`java.util.Enumeration` `getConditionalPermissionInfos`()

Returns the Conditional Permission Infos from the Conditional Permission

The returned Enumeration will return elements in the order they are kept in the Conditional Permission Table.

The Enumeration returned is based on a copy of the Conditional Permission Table and therefore will not throw exceptions if the Conditional Permission Table is changed during the course of reading elements from the Enumeration.

Returns:

An enumeration of the Conditional Permission Infos that are currently in the Conditional Permission Table.

5.5.4.4 *getConditionalPermissionInfo*

[ConditionalPermissionInfo](#) `getConditionalPermissionInfo(java.lang.String name)`

Return the Conditional Permission Info with the specified name.

Parameters:

name - The name of the Conditional Permission Info to be returned.

Returns:

The Conditional Permission Info with the specified name.

5.5.4.5 *getAccessControlContext*

`java.security.AccessControlContext`

`getAccessControlContext(java.lang.String[] signers)`

Returns the Access Control Context that corresponds to the specified signers.

Parameters:

signers - The signers for which to return an Access Control Context.

Returns:

An `AccessControlContext` that has the Permissions associated with the signer.

5.5.4.6 *createConditionalPermissionsUpdate*

[ConditionalPermissionsUpdate](#) `createConditionalPermissionsUpdate()`

Creates an update for the Conditional Permission Table. The update is a working copy of the current Conditional Permission Table. If the running Conditional Permission Table is modified before `commit` is called on the returned update, then the call to `commit` will fail. That is, the `commit` method will return `false` and no change will be made to the running Conditional Permission Table. There is no requirement that `commit` is eventually called on the returned update.

Returns:

An update for the Conditional Permission Table.

Since:

1.1

5.5.4.7 *createConditionalPermissionInfoBase*

[ConditionalPermissionInfoBase](#)

`createConditionalPermissionInfoBase(java.lang.String name,`

`ConditionInfo[] conditions,`

`PermissionInfo[] permissions,`

java.lang.String decision)

Creates a ConditionalPermissionInfoBase with the specified fields.

Parameters:

name - The name of the created ConditionalPermissionInfoBase or `null` to have a unique name generated when the created ConditionalPermissionInfoBase is committed in an update to the Conditional Permission Table.

conditions - The Conditions that need to be satisfied to enable the corresponding Permissions.

permissions - The Permissions that are enabled when the corresponding Conditions are satisfied.

decision - One of the following values:

- [allow](#)
- [deny](#)

Returns:

A ConditionalPermissionInfoBase object suitable for insertion in a [ConditionalPermissionsUpdate](#).

Throws:

java.lang.IllegalArgumentException - If the decision string is invalid.

Since:

1.1

[Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.5.5 org.osgi.service.condpermadmin

Interface ConditionalPermissionInfo

All Superinterfaces:

[ConditionalPermissionInfoBase](#)

```
public interface ConditionalPermissionInfo
```

```
extends ConditionalPermissionInfoBase
```

A binding of a set of Conditions to a set of Permissions. Instances of this interface are obtained from the Conditional Permission Admin service.

Version:

\$Revision: 5188 \$

Field Summary

Fields inherited from interface org.osgi.service.condpermadmin.[ConditionalPermissionInfoBase](#)

[ALLOW](#), [DENY](#)

Method Summary

void [delete\(\)](#)

Removes this Conditional Permission Info from the Conditional Permission Table.

Methods inherited from interface `org.osgi.service.condpermadmin.ConditionalPermissionInfoBase`

[getConditionInfos](#), [getGrantDecision](#), [getName](#), [getPermissionInfos](#)

Method Detail

5.5.5.1 delete

void `delete()`

Removes this Conditional Permission Info from the Conditional Permission Table.

Since this method changes the underlying permission table any [ConditionalPermissionsUpdates](#) that were created prior to calling this method can no longer be committed.

Throws:

`java.lang.SecurityException` - If the caller does not have `AllPermission`.

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.5.6 org.osgi.service.condpermadmin

Interface `ConditionalPermissionInfoBase`

All Known Subinterfaces:

[ConditionalPermissionInfo](#)

public interface `ConditionalPermissionInfoBase`

A binding of a set of Conditions to a set of Permissions. Instances of this interface are obtained from the Conditional Permission Admin service.

Since:

1.1

Version:

\$Revision: 5185 \$

Field Summary

static java.lang.String	ALLOW This string is used to indicate that a row in the conditional permission admin table should return a grant decision of ALLOW if the conditions are all satisfied and at least one of the permissions is implied.
static java.lang.String	DENY This string is used to indicate that a row in the conditional permission admin table should return a grant decision of DENY if the conditions are all satisfied and at least one of the permissions is implied.

Method Summary

ConditionInfo[]	getConditionInfos() Returns the Condition Infos for the Conditions that must be satisfied to enable the Permissions.
java.lang.String	getGrantDecision() Returns the grant decision for this Conditional Permission Info.
java.lang.String	getName() Returns the name of this Conditional Permission Info.
PermissionInfo[]	getPermissionInfos() Returns the Permission Infos for the Permission in this Conditional Permission Info.

Field Detail

5.5.6.1 ALLOW

static final java.lang.String **ALLOW**

This string is used to indicate that a row in the conditional permission admin table should return a grant decision of ALLOW if the conditions are all satisfied and at least one of the permissions is implied.

See Also:

[Constant Field Values](#)

5.5.6.2 DENY

static final java.lang.String **DENY**

This string is used to indicate that a row in the conditional permission admin table should return a grant decision of DENY if the conditions are all satisfied and at least one of the permissions is implied.

See Also:

[Constant Field Values](#)

Method Detail

5.5.6.3 getConditionInfos

[ConditionInfo](#)[] **getConditionInfos**()

Returns the Condition Infos for the Conditions that must be satisfied to enable the Permissions.

Returns:

The Condition Infos for the Conditions in this Conditional Permission Info.

5.5.6.4 *getPermissionInfos*

PermissionInfo[] **getPermissionInfos**()

Returns the Permission Infos for the Permission in this Conditional Permission Info.

Returns:

The Permission Infos for the Permission in this Conditional Permission Info.

5.5.6.5 *getGrantDecision*

java.lang.String **getGrantDecision**()

Returns the grant decision for this Conditional Permission Info.

Returns:

One of the following values:

- [allow](#) - The grant decision is allow.
 - [deny](#) - The grant decision is DENY.
-

5.5.6.6 *getName*

java.lang.String **getName**()

Returns the name of this Conditional Permission Info.

Returns:

The name of this Conditional Permission Info.

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.5.7 org.osgi.service.condpermadmin Interface ConditionalPermissionsUpdate

public interface **ConditionalPermissionsUpdate**

Update the Conditional Permission Table. There may be many update objects in the system at one time. If commit is called and the Conditional Permission Table has been modified since this update was created, then the call to commit will fail and this object should be discarded.

Since:

1.1

Version:

\$Revision: 5188 \$

Method Summary

boolean	commit() Commit the update.
Java.util.List	getConditionalPermissionInfoBases() This method returns the list of ConditionalPermissionInfoBases for this update.

Method Detail

5.5.7.1 *getConditionalPermissionInfoBases*

```
java.util.List getConditionalPermissionInfoBases()
```

This method returns the list of [ConditionalPermissionInfoBase](#)s for this update. This list is originally based on the Conditional Permission Table at the time this update was created. The list returned by this method will be replace the Conditional Permission Table if commit is called and is successful.

The elements of the list must NOT be instances of type [ConditionalPermissionInfo](#), but must rather be of type [ConditionalPermissionInfoBase](#). This is to ensure the [delete](#) method cannot be mistakenly used.

The list returned by this method is ordered and the most significant table entry is the first entry in the list.

Returns:

A List of the Conditional Permission Info Bases which represent the Conditional Permissions maintained by this update. Modifications to this list will not affect the Conditional Permission Table until successfully committed. The elements in this list must be of type [ConditionalPermissionInfoBase](#). The list may be empty if the Conditional Permission Table was empty when this update was created.

5.5.7.2 *commit*

```
boolean commit()
```

Commit the update. If no changes have been made to the Conditional Permission Table since this update was created, then this method will replace the Conditional Permission Table with this update's Conditional Permissions. This method may only be successfully called once on this object.

If any of the [ConditionalPermissionInfoBase](#) objects in the update list has null as a name it will be replaced with a [ConditionalPermissionInfoBase](#) object that has a generated name which is unique within the list.

No two entries in this update's Conditional Permissions may have the same name. Other consistency checks may also be performed. If the update's Conditional Permissions are determined to be inconsistent in some way then an `IllegalStateException` will be thrown.

This method returns `false` if the Conditional Permission Table has been modified since the creation of this update.

Returns:

`true` if the commit was successful. `false` if the Conditional Permission Table has been modified since the creation of this update.

Throws:

`java.lang.SecurityException` - If the caller does not have `AllPermission`.

`java.lang.IllegalStateException` - If the update's Conditional Permissions are not valid or inconsistent. For example, if this update has two Conditional Permissions in it with the same name, then this exception will be thrown.

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#)

DETAIL: FIELD | CONSTR | [METHOD](#)

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: NESTED | FIELD | [CONSTR](#) | [METHOD](#)

DETAIL: FIELD | [CONSTR](#) | [METHOD](#)

5.5.8 org.osgi.service.condpermadmin Class ConditionInfo

`java.lang.Object`

`org.osgi.service.condpermadmin.ConditionInfo`

```
public class ConditionInfo
extends java.lang.Object
```

Condition representation used by the Conditional Permission Admin service.

This class encapsulates two pieces of information: a *Condition type* (class name), which must implement `Condition`, and the arguments passed to its constructor.

In order for a `Condition` represented by a `ConditionInfo` to be instantiated and considered during a permission check, its `Condition` class must be available from the system classpath.

The `Condition` class must either:

- Declare a public static `getCondition` method that takes a `Bundle` object and a `ConditionInfo` object as arguments. That method must return an object that implements the `Condition` interface.
- Implement the `Condition` interface and define a public constructor that takes a `Bundle` object and a `ConditionInfo` object as arguments.

Version:

\$Revision: 5183 \$

Constructor Summary

[ConditionInfo](#)(java.lang.String encodedCondition)

Constructs a ConditionInfo object from the specified encoded ConditionInfo string.

[ConditionInfo](#)(java.lang.String type, java.lang.String[] args)

Constructs a ConditionInfo from the specified type and args.

Method Summary

boolean	equals (java.lang.Object obj) Determines the equality of two ConditionInfo objects.
Java.lang.String[]	getArgs () Returns arguments of this ConditionInfo.
java.lang.String	getEncoded () Returns the string encoding of this ConditionInfo in a form suitable for restoring this ConditionInfo.
java.lang.String	getType () Returns the fully qualified class name of the condition represented by this ConditionInfo.
int	hashCode () Returns the hash code value for this object.
java.lang.String	toString () Returns the string representation of this ConditionInfo.

Methods inherited from class java.lang.Object

Clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

5.5.8.1 ConditionInfo

```
public ConditionInfo(java.lang.String type,
                    java.lang.String[] args)
```

Constructs a ConditionInfo from the specified type and args.

Parameters:

type - The fully qualified class name of the Condition represented by this ConditionInfo.

args - The arguments for the Condition. These arguments are available to the newly created Condition by calling the [getArgs\(\)](#) method.

Throws:

java.lang.NullPointerException - If type is null.

5.5.8.2 *ConditionInfo*

```
public ConditionInfo(java.lang.String encodedCondition)
```

Constructs a `ConditionInfo` object from the specified encoded `ConditionInfo` string. White space in the encoded `ConditionInfo` string is ignored.

Parameters:

`encodedCondition` - The encoded `ConditionInfo`.

Throws:

`java.lang.IllegalArgumentException` - If the `encodedCondition` is not properly formatted.

See Also:

[getEncoded\(\)](#)

Method Detail

5.5.8.3 *getEncoded*

```
public final java.lang.String getEncoded()
```

Returns the string encoding of this `ConditionInfo` in a form suitable for restoring this `ConditionInfo`.

The encoding format is:

```
[type "arg0" "arg1" ...]
```

where *argN* are strings that are encoded for proper parsing. Specifically, the `"`, `\`, carriage return, and line feed characters are escaped using `\`, `\\`, `\r`, and `\n`, respectively.

The encoded string contains no leading or trailing whitespace characters. A single space character is used between *type* and `"arg0"` and between the arguments.

Returns:

The string encoding of this `ConditionInfo`.

5.5.8.4 *toString*

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

Returns the string representation of this `ConditionInfo`. The string is created by calling the `getEncoded` method on this `ConditionInfo`.

Overrides:

`toString` in class `java.lang.Object`

Returns:

The string representation of this `ConditionInfo`.

5.5.8.5 *getType*

```
public final java.lang.String getType()
```

Returns the fully qualified class name of the condition represented by this `ConditionInfo`.

Returns:

The fully qualified class name of the condition represented by this `ConditionInfo`.

5.5.8.6 *getArgs*

```
public final java.lang.String[] getArgs()
```

Returns arguments of this `ConditionInfo`.

Returns:

The arguments of this `ConditionInfo`. An empty array is returned if the `ConditionInfo` has no arguments.

5.5.8.7 *equals*

```
public boolean equals(java.lang.Object obj)
```

Determines the equality of two `ConditionInfo` objects. This method checks that specified object has the same type and args as this `ConditionInfo` object.

Overrides:

`equals` in class `java.lang.Object`

Parameters:

`obj` - The object to test for equality with this `ConditionInfo` object.

Returns:

true if `obj` is a `ConditionInfo`, and has the same type and args as this `ConditionInfo` object; false otherwise.

5.5.8.8 *hashCode*

```
public int hashCode()
```

Returns the hash code value for this object.

Overrides:

`hashCode` in class `java.lang.Object`

Returns:

A hash code value for this object.

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.6 Open Issues

5.7 Closed Issues

5.7.1 Default Decision

5.7.1.1 *Original Issue*

In a system where denying permissions is explicitly and appropriately handled, it is useful to be able to control the decision if no-one has an explicit decision. Currently if no row in the table matches, then the decision taken is to disallow the requested permission. This behavior is fine in a system where rights cannot be denied as well as

granted. Since this system allows for permissions to be denied it becomes useful (and would lead to fewer table entries) if the decision taken when no-one has made an explicit decision could be set to allow the request.

One common use-case where this is important is when someone wishes to have the last row in the table turn on all of the permissions. The below table is an example:

Name	Conditions	Permissions	Decision
R1	<whatever>	<whatever>	DENY
LastRow	<empty>	java.lang.AllPermission	ALLOW

In this use case, the last row in the table allows all permissions to everyone. This makes the security of the system based on what is denied as opposed to what is granted. However, many of the API (in particular the legacy API) have specified that they add elements to the end of the table. Also, it is very easy to add items to the end of a list, whereas adding things next-to-the end is somewhat trickier. Therefore, it would make the programming model easier if instead there were a “default” decision that was taken if no rows of the table matched. If people knew that the default decision to be taken was ALLOW as opposed to abstain, they could safely add their entries to the end of the list, rather than having to add a row that must always remain at the end of the list.

This enhancement has not been specified pending further discussion of the merits of the proposal.

5.7.1.2 Resolution

Despite the difficulties introduced to the programmer when using an ordered table this enhancement will not be done. The javadoc for the original API that added items to the table have been modified to specify that they will add their entries to the beginning of the table rather than the end so that it might be easier to use the old API in conjunction with the Update API.

5.7.2 An Alternate Update model

5.7.2.1 Original Issue

Another model for atomic updates is one in which the “createUpdate” method can create multiple updates and that only the “commit” method need be atomic. In this mode, each update gets a working copy of the Permission Admin Table and when commit happens successfully its copy becomes the new Permission Admin Table. If multiple Updates are based off of the same conceptual table generation number then only one of the updates would succeed at commit time. All of the other updates would fail at commit time, with an explicit exception being thrown indicating that the update was based on an older generation number. It would be up to the code at that point to re-do the work it had done.

This option is viable and has the advantage that blocking semantics and timeouts need not be specified. It however suffers the drawback that the error case will happen even in properly written code and thus will need to be dealt with by the client code. In both proposals the client code needs to deal with the problems of inconsistent lists and other semantic issues, but in this proposal the code would also have to deal with the hirsute problem of a lost race.

Since the existing proposal seems to be easier to use from a client perspective it has been chosen, pending further discussion of the merits of the multiple-update proposal.

5.7.2.2 Resolution

This model has been adopted in this version of the specification. Version 0.4 of this document proposed a mechanism whereby an exclusive lock could be held, ensuring multiple simultaneous updates could not occur. In this solution there is no supplied methodology to ensure races do not occur. Lost races are reported to the user, who is responsible for redoing the work lost.

5.7.3 Utility methods

5.7.3.1 Original Issue

The methods `createInfoBase()` and `generateUniqueName()` are currently specified on the `org.osgi.service.condpermadmin.ConditionalPermissionAdminUpdate` object. However, these methods have nothing to do with the particular update and would be better served as static methods on some Utility class. There does not seem to be any generic "Utility" class in the OSGi framework. It would be useful to have such a class for methods like these to be specified.

Since this RFC does not appear to be the proper place to propose such a Utility class the methods will remain where they are, pending further discussion.

5.7.3.2 Resolution

Since Java has a deficiency in supporting static style methods, these methods will remain on an interface. However, they have been moved from the Update interface to the Admin interface.

6 Considered Alternatives

The idea here is to have a "DeniablePermission" class that extends `BasicPermission`. All of the classes listed above that currently extend `BasicPermission` would be changed to extend `DeniablePermission`.

This solution has the advantage of being very easy to implement. However, there are subtleties of the syntax which are not desirable. For example, if someone were to specify `"*-a.b"` and also `"*-a.c"` resolves to `"**"`. Instead, the user who wanted to restrict both `a.b` and `a.c` should have said `"*-a.b,a.c"`. This might confuse people and make it difficult to read and understand the security constraints placed on the `Permission`.

The javadoc for the `DeniablePermission` can be found below:

6.1 `com.bea.sandbox.security.permission`

Class `DeniablePermission`

[java.lang.Object](#)

└ [java.security.Permission](#)

└ [java.security.BasicPermission](#)

└ `com.bea.sandbox.security.permission.DeniablePermission`

All Implemented Interfaces:

[Serializable](#), [Guard](#)

```
public class DeniablePermission
extends BasicPermission
```

This is an extension of BasicPermission which understands denials as well as grants.

The name for a DeniablePermission has two parts, the part before the '-' character, and the part after the '-' character. If there is no '-' character in the name then DeniablePermission acts exactly like a BasicPermission. The first part of the name is the Permission granted, while the second part of the name are the things denied.

The rules for the first part of the name (things granted) follow the same rules as BasicPermission. The first part of the name for a DeniablePermission is the name of the given permission (for example, "exit", "setFactory", "print.queueJob", etc). The naming convention follows the hierarchical property naming convention. An asterisk may appear by itself, or if immediately preceded by a "." may appear at the end of the name, to signify a wildcard match. For example, "*" and "java.*" are valid, while "*java", "a*b", and "java*" are not valid.

The rules for the second part of the name (things denied) are as follows. The second part of the name consists of comma separated resource names that should be denied access. The individual resource names are called denial strings. These denial strings may be individually listed resources ("exit", "setFactory", "print.queueJob", etc). The individual denied strings may end with ".*" in order to signify a wildcard match. Examples of valid names with denial strings are

- "com.acme.*-com.acme.security.*"
- "*-com.acme.accounting.*,com.acme.cfo.getMail"

There are other rules about DeniablePermission names:

- A name may not have a '-' character if the grant portion of the name is not a wildcard
- A denial string may not be ""
- A denial string may not have an embedded ""
- The last character of a name may not be '-'

See Also:

[Serialized Form](#)

Constructor Summary

[DeniablePermission](#)([String](#) name)

This cuts off anything prior to the '-' character for the super

[DeniablePermission](#)([String](#) name,

[String](#) actions)

This cuts off anything prior to the '-' character for the super

Method Summary

boolean	equals (Object cmp)
int	hashCode ()
boolean	implies (Permission p)
String	toString ()

Methods inherited from class [java.security.BasicPermission](#)

[getActions](#), [newPermissionCollection](#)

Methods inherited from class [java.security.Permission](#)

[checkGuard](#), [getName](#)

Methods inherited from class [java.lang.Object](#)

[clone](#), [finalize](#), [getClass](#), [notify](#), [notifyAll](#), [wait](#), [wait](#), [wait](#)

Constructor Detail

6.1.1 DeniablePermission

```
public DeniablePermission(String name)
```

This cuts off anything prior to the '-' character for the super

Parameters:

name - The name with the optional '-' character

6.1.2 DeniablePermission

```
public DeniablePermission(String name,  
                           String actions)
```

This cuts off anything prior to the '-' character for the super

Parameters:

name - The name with the optional '-' character

actions - The actions

Method Detail

6.1.3 implies

```
public boolean implies(Permission p)
```

Overrides:

[implies](#) in class [BasicPermission](#)

See Also:

[BasicPermission.implies\(java.security.Permission\)](#)

6.1.4 hashCode

```
public int hashCode()
```

Overrides:

[hashCode](#) in class [BasicPermission](#)

See Also:

[BasicPermission.hashCode\(\)](#)

6.1.5 equals

```
public boolean equals(Object cmp)
```

Overrides:

[equals](#) in class [BasicPermission](#)

See Also:

[BasicPermission.equals\(java.lang.Object\)](#)

6.1.6 toString

```
public String toString()
```

Overrides:

[toString](#) in class [Permission](#)

<http://www-beace/beace-site/5.0/com.bea.sandbox.dp/javadoc/com/bea/sandbox/security/permission/DeniablePermission.html> - skip-navbar bottom#skip-navbar bottom

Package **Class** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS [FRAMES](#) [NO FRAMES](#) [All Classes](#) [All Classes](#)

SUMMARY: NESTED | FIELD | [CONSTR](#) | [METHOD](#) DETAIL: FIELD | [CONSTR](#) | [METHOD](#)

Documentation is available at <http://docs-stage/msa/docs30>
Copyright 2007 BEA Systems Inc.

6.2 Deny Permission Column

In this solution a deny permission column was added to the tuple table. There are several reasons this solution was reject. This solution did not handle certain important use cases (which will be detailed in section 6.2.10). Furthermore it is difficult to optimize this solution since all rows of the table must be evaluated even if one of the rows has returned an “allows” result.

The following are the discussions and requirements of that solution.

6.2.1 Add Deny Column to Permission Table Discussion

The first part of the solution is to take the security table which currently has three columns and add a fourth column. Currently, the three columns would be labeled “name”, “condition” and “permission”. In this solution the columns of the table would be labeled “name”, “condition”, “allows permission” and “denies permission”.

Today a “permission” cell in the table contains a list of Permission objects. If any of these objects “imply” the permission being sought then the result of the security check is to allow the operation. Hence, the individual permissions act like a logical “OR” operation, since if *any* of the Permission objects in the cell return “true” then the security check returns “true” (or ALLOW). Note that at least one Permission object must “imply” the

permission. Hence an empty permission set in the “permission” column of the table would always return “false” (or DENY).

In this solution the “allows permission” cell behaves in the same way as above, and would return the same “true” (ALLOW) or “false” (ABSTAIN) decision as it does currently. However, after the “allow permission” cell has been consulted the “denies permission” cell will also be checked. The “denies permission” cell must return “false” in order for the security check to pass. In other words the security check result can be gotten using the following pseudo-formula:

Security-check-result = (allow-permission-result) && !(denies-permission-result).

The “denies-permission” cells contains a list of Permission objects, just as the “allows-permission” cells do. The individual permissions in the “denies-permission” cells are OR’d together to get the result. Therefore in order for the whole security check to pass ALL of the Permissions in the “denies-permission” cell must NOT imply the Permission being asked for.

A simple example should make this clear. Assume I wish to grant access to all packages except the “com.acme.secret” package and the “com.acme.sauce” package. Here is a row of cells in the security table:

Name	Condition	Allow Permission	Deny Permission
Example	<empty> (always true)	Package(“**”)	Package(“com.acme.secret”), Package(“com.acme.sauce”)

If a bundle is trying to access the package “com.acme.transaction” they would first check against Package(“**”) which would return “true”. Then the system would check both Package(“com.acme.secret”) which would return “false” and Package(“com.acme.sauce”) which would return “false”, and so the “Deny Permission” cell would have a value of “false”. The security check: (allow-permission-result) && !(denies-permission-result) would end up being (true) && !(false) and would hence be “true” and access to the “com.acme.transaction” package would be allowed.

On the other hand, if a bundle is trying to access the package “com.acme.secret” they would first check against Package(“**”) which would return “true”. Then the system would check Package(“com.acme.secret”) which would return “true”, and so the “Deny Permission” cell would have a value of “true”. The security check (allow-permission-result) && !(denies-permission-result) would end up being (true) && !(true) and would hence be “false” and access to the “com.acme.secret” package would be denied.

Some things about this proposed solution:

1. The way the table works today if the permission succeeds against ANY row in the table then the check will succeed. Hence you could have a scenario where something is denied in one row of the table but succeeds in another row of the table. According to the current rules that would allow access to that permission. Specifying a separate “deny” table (rather than adding a column to the existing table) has been considered.
2. In the condition column an empty cell implies “true”. However, in the “Deny Permission” column it appears that an empty cell should imply “false”.
3. For this RFC to be complete new API and a new “normalized” file format will need to be defined.

6.2.1.1 Deny Column Requirements

A fourth column shall be added to the conceptual permission table. For the purposes of these requirements the existing column in the tuple table will be called the “allow permissions” column while the new column being added will be called the “deny permissions” column.

The “deny permission” column of the table shall contain a list of PermissionInfo objects. A target permission shall be implied when the following are true:

1. At least one permission in the “allow permission” column has a permission that implies the target permission
2. None of the permissions in the “deny permission” column has a permission the implies the target permission

If there are no permissions in the “deny permission” column then the column naturally returns “false” and the result of the permission check comes solely from the permissions in the “allow permission” column.

The “deny permission” column shall be consulted when determining if a tuple should be postponed. Therefore the boxes in Figure 9.42 of the Conditional Permission Admin Specification that ask if a permission implies P shall take the “deny permission” column into account as described above.

However, due to these requirements it is no longer the case that any tuple which eventually returns “true” means that the permission check can succeed. All of the tuples whose conditions match and who have at least one entry in the “deny permissions” column shall be checked in order to ensure that no denial is present.

6.2.2 NOT Condition Discussion

Using the above specified mechanism it is possible to deny access to resources. While this achieves the requirements to deny resources there is still something missing. For example, suppose we wanted to allow bundles signed by the ACME Company to have access to all packages but we wanted all other bundles in the system to not have access to the “com.acme.secret” and “com.acme.sauce” packages. There is no way using standard OSGi conditions to allow this to happen. Instead, what is needed is a NOT boolean condition operator.

Consider the use case where you want ACME (the provider of the system software) to have access to all packages but everyone else should not be allowed access to the “com.acme.secret” and “com.acme.sauce” packages. In this case, you would need the Not condition. Your permission table would look something like this:

Name	Condition	Allows Permission	Deny Permission
ACME	Signed By ACME	Package(“**”)	<empty>
Non-System	NOT Signed By ACME	Package(“**”)	Package(“com.acme.secret”) Package(“com.acme.sauce”)

In the above table the bundles signed by ACME will have access to all packages, while all bundles NOT signed by Acme can access neither the “com.acme.secret” package nor the “com.acme.sauce” package.

Another example will illustrate how you could allow multiple providers of system software to use the deny permission feature. For example, the ACME Corporation may be using Spring to provide dependency injection. In this example, both ACME and Spring need to have access to all packages, but no-one else should be granted

access to the same restricted packages as above. In this case, your permission table would look something like this:

Name	Condition	AllowsPermission	DenyPermission
ACME	Signed By ACME	Package("**")	<empty>
Spring	Signed By Spring	Package("**")	<empty>
Non-System	NOT Signed By ACME	Package("**")	Package("com.acme.secret")
	NOT Signed By Spring		Package("com.acme.sauce")

Since the Conditions in a single tuple row must ALL satisfy the condition the above table would achieve the desired result or allowing the system bundles (signed by Acme and Spring) to have access to all packages while denying access to the "com.acme.secret" and "com.acme.sauce" packages to all non-system bundles.

The interesting thing about the above examples is they can both be simplified with the following table:

Name	Condition	Allow Permission	Deny Permission
Default	<empty>	Package("**")	<empty>
Non-System	NOT Signed By ACME	<empty>	Package("com.acme.secret")
	NOT Signed by Spring		Package("com.acme.sauce")

The above table will achieve the same results as the previous table with fewer entries. The system may choose to optimize these sorts of conditions.

6.2.3 NotCondition Requirements

A new class named "org.osgi.service.condperadmin.NotCondition" shall be added to the framework. The javadoc shall be considered a binding part of this specification, and is found in the next sub-section.

In particular, the NotCondition shall have another condition upon which it bases its return values. This base condition will be called the "base condition." The NotCondition shall return the same values as the base for the mutable and postponed properties and shall return the logical NOT of the base satisfied property.

Currently the system understands the static form:

```
static Condition getCondition(Bundle bundle, String parameters);
```

However, the NotCondition shall have the static factory method:

```
static Condition getCondition(Bundle bundle, Condition base);
```

The Encoded ConditionInfo string shall be enhanced to understand the "!" character at the start of the condition. If the "!" character is found the base condition shall be constructed as before, but it will be used as the base condition of the NotCondition. The encoded string shall have the form:

“[+ “!”* + fully-qualified-condition-class + “ \” + argument-string + “\”]”

Note that the “!” character is optional (denoted by the *).

Some examples of the encoded ConditionInfo string might include:

[!org.osgi.service.condpermadmin.BundleSignerCondition “* ; cn=Whatever, o=ACME, c=US”]

[!org.osgi.service.condpermadmin.BundleLocationCondition “/home/acme/foo/b.jar”]

6.2.4 NotCondition Javadoc

C:\tmp\osgi\rfc120\org\osgi\service\condpermadmin\NotCondition.html - skip-navbar_top#skip-navbar_top

[Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.osgi.service.condpermadmin

Class NotCondition

java.lang.Object

└─ org.osgi.service.condpermadmin.NotCondition

```
public class NotCondition
```

```
extends java.lang.Object
```

This is a condition that is formed as the logical NOT operation of another condition.

Method Summary

static Condition	getCondition (Bundle bundle, Condition condition)
	Creates a condition that returns the logical NOT of another condition.

Methods inherited from class java.lang.Object

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

Method Detail

getCondition

```
public static Condition getCondition(Bundle bundle,
                                    Condition condition)
```

This uses the Condition form as proposed in RFC 120

Parameters:

bundle - The bundle for which this condition is being created

condition - The condition to "NOT"

Returns:

A condition that will have the same mutable and postponed values as the base condition but which will return the logical NOT of the satisfied value of the base condition.

C:\tmp\osgi\rfc120\org\osgi\service\condperadmin\NotCondition.html - skip-navbar_bottom#skip-navbar_bottom

[Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#) [All Classes](#)

[SUMMARY: NESTED | FIELD | CONSTR | METHOD](#)

[DETAIL: FIELD | CONSTR | METHOD](#)

6.2.5 Other API modifications

In order to fully support the modifications described in sections 6.2.1 and 6.2.2 some existing API's will also need to change.

The following sections will give javadoc for the new API and constructors and should be considered a binding part of this specification.

6.2.6 org.osgi.service.condperadmin.ConditionInfo

The following API shall be added to the org.osgi.service.condperadmin.ConditionInfo class:

6.2.6.1 Constructor

ConditionInfo

```
public ConditionInfo(boolean not,  
                    java.lang.String type,  
                    java.lang.String[] args)
```

Constructs a ConditionInfo from the specified type and args.

Parameters:

not - If true, the condition described should return the associated NotCondition, using the type and args as the base Condition.

type - The fully qualified class name of the Condition represented by this ConditionInfo.

args - The arguments for the Condition. These arguments are available to the newly created Condition by calling the [getArgs\(\)](#) method.

Throws:

java.lang.NullPointerException - If type is null.

6.2.6.2 isNot method

isNot

```
public boolean isNot()
```

Returns true if this ConditionInfo describes a NotCondition

Returns:

true if this ConditionInfo describes a NotCondition

6.2.7 org.osgi.service.condperadmin.ConditionPermissionInfo

The following API shall be added to the org.osgi.service.condperadmin.ConditionPermissionInfo interface.

6.2.7.1 *getDenyPermissionInfos* method

getDenyPermissionInfos

`PermissionInfo[] getDenyPermissionInfos()`

Returns the Permission Infos for the Deny Permission column in this Condition Permission Info.

Returns:

The Deny Permission Infos for the Permission in this Conditional Permission Info.

6.2.8 **org.osgi.service.condpermadmin.ConditionalPermissionAdmin**

The following API shall be added to the `org.osgi.service.condpermadmin.ConditionalPermissionAdmin` interface.

6.2.8.1 *addConditionalPermissionInfo*

addConditionalPermissionInfo

[ConditionalPermissionInfo](#) `addConditionalPermissionInfo(ConditionInfo[] conds, PermissionInfo[] perms, PermissionInfo[] denyPerms)`

Create a new Conditional Permission Info. The Conditional Permission Info will be given a unique, never reused name.

Parameters:

`conds` - The Conditions that need to be satisfied to enable the corresponding Permissions.

`perms` - The Permissions that are enable when the corresponding Conditions are satisfied.

`denyPerms` - The Permissions that are denied when the corresponding Conditions are satisfied.

Returns:

The `ConditionalPermissionInfo` for the specified Conditions and Permissions.

Throws:

`java.lang.SecurityException` - If the caller does not have `AllPermission`.

6.2.8.2 *setConditionalPermissionInfo*

setConditionalPermissionInfo

[ConditionalPermissionInfo](#) `setConditionalPermissionInfo(java.lang.String name, ConditionInfo[] conds, PermissionInfo[] perms, PermissionInfo[] denyPerms)`

Set or create a Conditional Permission Info with a specified name. If the specified name is `null`, a new Conditional Permission Info must be created and will be given a unique, never reused name. If there is currently no Conditional Permission Info with the specified name, a new Conditional Permission Info must be created with the specified name. Otherwise, the Conditional Permission Info with the specified name must be updated with the specified Conditions and Permissions.

Parameters:

`name` - The name of the Conditional Permission Info, or `null`.

`conds` - The Conditions that need to be satisfied to enable the corresponding Permissions.

`perms` - The Permissions that are enable when the corresponding Conditions are satisfied.

`denyPerms` - The Permissions that are denied when the corresponding Conditions are satisfied.

Returns:

The `ConditionalPermissionInfo` that for the specified name, Conditions and Permissions.

Throws:

`java.lang.SecurityException` - If the caller does not have `AllPermission`.

6.2.9 Issues

6.2.10 Friends API

One use case (which is not in the original set of use cases) that is not handled cleanly by the above solution is the friends use case. In this use case we want to express the following scenario:

4. Pepsi wants to deny permissions to com.pepsi.* for everyone but Pepsi
5. Pepsi wants to allow Coke to have permission to com.pepsi.z.*

Using the existing scheme you cannot express the above set of requirements simply. The best you could do would be to have something like this:

Name	Condition	Allow Permissions	Deny Permissions
N1	! Signed by Pepsi ! Signed by Coke	Package(*)	Package("com.pepsi.*")
N2	Signed by Coke	Package(*)	Package("com.pepsi.a.*") Package("com.pepsi.b.*") ... Package("com.pepsi.y.*")
N3	<empty>	Package(*)	<empty>

Notice that in the N2 row above that it denies access to Coke to every package **except** the com.pepsi.z.* package. This is brittle and error prone, since with every new package hierarchy (perhaps com.pepsi.aa.*) that Pepsi might add to their system a new Permission would have to be added in the N2 line. It is this sort of complexity that this solution was attempting to address.

Given that this not one of the original use-cases and there does not appear to be a satisfactory solution to this problem this specification will not address this use case.

7 Security Considerations

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

8 Document Support

8.1 References

- [1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
- [2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0
- [3]. [RFP-78 Security Use Cases](#)

8.2 Author's Address

Name	John Wells
Company	BEA Systems, Inc.
Address	150 Allen Road, Liberty Corner, NJ
Voice	(908) 580-3127
e-mail	jwells@bea.com

8.3 Acronyms and Abbreviations

8.4 End of Document



RFC 121 Bundle Tracker

Draft

17 Pages

Abstract

The BundleTracker class simplifies tracking bundles much like the ServiceTracker simplified tracking services.

Copyright © IBM Corporation 2007.

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	3
0.3 Revision History	3
1 Introduction.....	3
2 Application Domain.....	4
3 Problem Description	4
4 Requirements.....	4
5 Technical Solution.....	5
5.1 Design Discussion	5
5.1.1 Overview	5
5.1.2 org.osgi.util.tracker package	5
5.1.3 Tracking Criteria.....	6
5.1.4 Synchronous Listener	6
5.1.5 Customized object	7
5.2 org.osgi.util.tracker.BundleTracker	Error! Bookmark not defined.
5.2.1 context.....	Error! Bookmark not defined.
5.2.2 BundleTracker.....	Error! Bookmark not defined.
5.2.3 addingBundle	Error! Bookmark not defined.
5.2.4 close.....	Error! Bookmark not defined.
5.2.5 getBundles	Error! Bookmark not defined.
5.2.6 getObject.....	Error! Bookmark not defined.
5.2.7 getTrackingCount.....	Error! Bookmark not defined.
5.2.8 modifiedBundle	Error! Bookmark not defined.
5.2.9 open	Error! Bookmark not defined.
5.2.10 remove	Error! Bookmark not defined.
5.2.11 removedBundle.....	Error! Bookmark not defined.
5.2.12 size.....	Error! Bookmark not defined.
5.3 org.osgi.util.tracker.BundleTrackerCustomizer.....	Error! Bookmark not defined.
5.3.1 addingBundle	Error! Bookmark not defined.
5.3.2 modifiedBundle	Error! Bookmark not defined.
5.3.3 removedBundle.....	Error! Bookmark not defined.

6 Considered Alternatives	14
6.1 Using Services to model Bundles	14
6.2 Using asynchronous Bundle Listener	15
7 Security Considerations	16
8 Document Support	17
8.1 References	17
8.2 Author's Address	17
8.3 Acronyms and Abbreviations	17
8.4 End of Document	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.

Revision	Date	Comments
Initial	7 August 2007	Initial Draft.
Draft 2	13 September 2007	Based upon CPEG discussion, I remove support for async BundleListener use by the tracker.
Draft 3	3 October 2007	Based upon CPEG discussion, I modified the customizer signature to pass the BundleEvent, if any, which triggered the action.

1 Introduction

Service Tracker[4] has long been around (since Release 2) and has long been a very useful tool providing a simple and correct way to track a set of services in the face of dynamism. It is very useful for implementing

whiteboard pattern approaches. With the advent of the extender model, it is now important to have a simple and correct way to track a set of bundles in the face of dynamism.

2 Application Domain

Any bundle, such as an extender bundle, that needs to track a set of bundles in a given range of states will need code to enable this tracking. Currently this is custom code for each such bundle.

3 Problem Description

Tracking bundles and services in the OSGi environment is challenging to do simply and correctly. Bundles may change state at any time and the bundle which needs to do the tracking will be started after a set of bundles already are present. The same challenges present for tracking services are also present for tracking bundles. Like the Service Tracker class introduced in Release 2, a Bundle Tracker class is needed[3] to define a standard, correct and easy-to-use way to track bundles.

4 Requirements

The following requirements are met by the proposed solution:

1. The Bundle Tracker class must be modeled along the Service Tracker class to provide a familiar pattern to developers.
2. The Bundle Tracker class must be in the `org.osgi.util.tracker` package to share code with Service Tracker reducing size, errors and maintenance.
3. The Bundle Tracker class must be correct with respect to the dynamic nature of OSGi.

4. The Bundle Tracker class must track all existing bundles which match the specified criteria as well as bundle whose state change to match the specified criteria after the tracker is opened.
5. The Bundle Tracker must be thread safe.
6. The Bundle Tracker must be able to support early versions of the framework (SynchronousBundleListener support is the minimal requirement.)

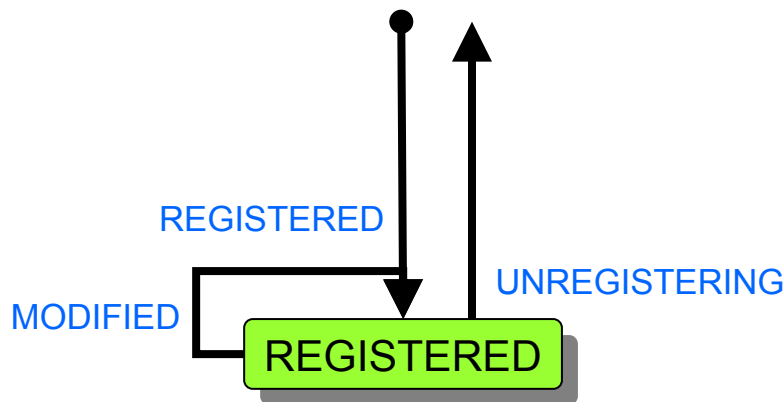
5 Technical Solution

5.1 Design Discussion

5.1.1 Overview

The Service Tracker design is based upon tracking service which are registered and match some specified criteria. Service events are used to indicate a state change in the service and are (only) synchronously delivered. During event processing, the event type along with criteria match against the service metadata is used to decide whether the service is to be added or removed from the tracker, or has just been modified.

The state diagram of a service is very simple.



The state diagram of a bundle is much more complex. The Bundle Tracker design is based upon the bundle's state. Bundles are tracked if they are in a set of states and not tracked otherwise. In this design, the state of the bundle is of primary importance and not the type of the bundle event received by the tracker.

5.1.2 org.osgi.util.tracker package

This design places the new Bundle Tracker into the existing org.osgi.util.tracker package. This is valuable for 2 reasons. First, it enables code sharing with the Service Tracker class. The tracking logic has been refactored from the Service Tracker class into an abstract base class which is then used by the Bundle Tracker. This reduces footprint and (hopefully) errors due to maintenance of duplicated code.

Given the name of the package, which does not include the word “service”, there is no package naming issue. However section 701 of the spec will need to be renamed from Service Tracker Specification to simply Tracker Specification.

Furthermore, with this addition to the package, the version of the package is incremented to 1.4.

5.1.3 Tracking Criteria

The tracking criteria for the Bundle Tracker are supplied as a bit mask in the constructor. This mask is an ORing of a set of bundle states. If a bundle is in one of those states, the Bundle Tracker will track it. Since the tracker must track bundles whose state matches the criteria at the time the tracker is opened as well as bundles whose state changes to match the criteria after the tracker is opened, a consistent test is needed.

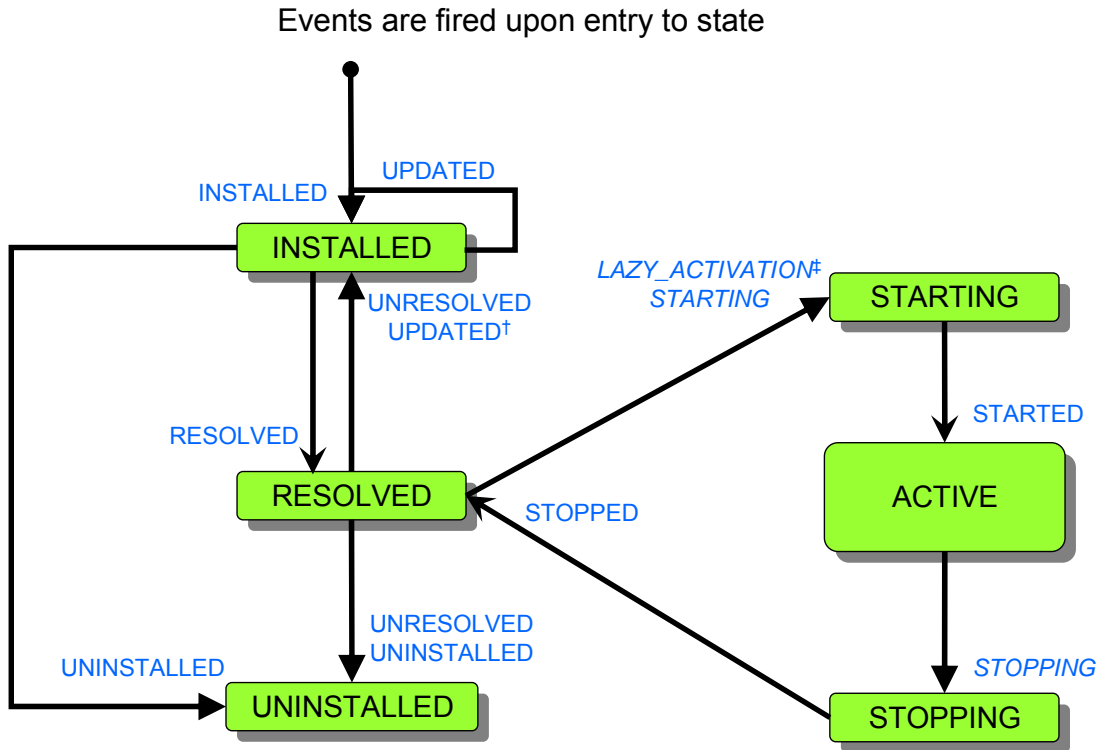
During tracker open, the only state to test is the bundle’s state as visible via `Bundle.getState`. However, while processing bundle events, both the bundle’s state and the event type are available for examination. Bundle event types are not sufficient to describe the resulting state of the bundle. The event type `UNRESOLVED` can be fired for entry to the `INSTALLED` and `UNINSTALLED` state.

In order to provide a simple and consistent test, the Bundle Tracker always examines the bundle’s state during open and bundle event processing. The bundle event type is not use. The delivery of the bundle event is used as a trigger to test the bundle against the tracking criteria to decide of the bundle should be tracked or untracked.

5.1.4 Synchronous Listener

The Bundle Tracker uses a Synchronous Bundle Listener. With synchronous bundle event processing, the bundle’s state is set before the event is synchronously fired and the event is delivered before the state can change again.

The following diagram depicts the bundle states and the event fired upon entry to those states. Note that entry to `STARTING` and `STOPPING` states is only signaled to synchronous bundle listeners and would not be reliably observable to asynchronous bundle listeners.



† only if updated

‡ only if lazy activation; STARTING is later fired when activation commences

Events in *italics* are only delivered to synchronous bundle listeners

5.1.5 Customized object

Like Service Tracker, the Bundle Tracker also allows the tracking of a customized object (object returned from BundleTrackerCustomized.addingBundle) along with the tracked bundle. For Service Tracker, this customized object is typically the service object. For Bundle Tracker, the default implementation of addingBundle simply returns the bundle.

5.2 org.osgi.util.tracker.BundleTracker

java.lang.Object

 org.osgi.util.tracker.BundleTracker

All Implemented Interfaces:

[BundleTrackerCustomizer](#)

```

public class BundleTracker
extends java.lang.Object
implements BundleTrackerCustomizer
  
```

The `BundleTracker` class simplifies tracking bundles much like the `ServiceTracker` simplifies tracking services.

A `BundleTracker` object is constructed with state criteria and a `BundleTrackerCustomizer` object. A `BundleTracker` object can use the `BundleTrackerCustomizer` object to select which bundles are tracked and to create a customized object to be tracked with the bundle. The `BundleTracker` object can then be opened to begin tracking all bundles whose state matches the specified state criteria.

The `getBundles` method can be called to get the `Bundle` objects of the bundles being tracked. The `getCustomizedObject` method can be called to get the customized object for a tracked bundle.

The `BundleTracker` class is thread-safe. It does not call a `BundleTrackerCustomizer` object while holding any locks. `BundleTrackerCustomizer` implementations must also be thread-safe.

Since:
1.4

Field Summary

<code>protected</code> <code>org.osgi.framework.BundleContext</code>	<u>context</u> Bundle context this <code>BundleTracker</code> object is tracking against.
---	--

Constructor Summary

<u>BundleTracker</u> (<code>org.osgi.framework.BundleContext context</code> , <code>int stateMask</code> , <u>BundleTrackerCustomizer</u> customizer)	Create a <code>BundleTracker</code> object for bundles whose state is present in the specified state mask.
--	--

Method Summary

<code>java.lang.Object</code>	<u>addingBundle</u> (<code>org.osgi.framework.Bundle bundle</code> , <code>org.osgi.framework.BundleEvent event</code>) Default implementation of the <code>BundleTrackerCustomizer.addingBundle</code> method.
<code>void</code>	<u>close</u> () Close this <code>BundleTracker</code> object.
<code>org.osgi.framework.Bundle[]</code>	<u>getBundles</u> () Return an array of <code>Bundle</code> objects for all bundles being tracked by this <code>BundleTracker</code> object.
<code>java.lang.Object</code>	<u>getObject</u> (<code>org.osgi.framework.Bundle bundle</code>) Returns the customized object for the specified <code>Bundle</code> object if the bundle is being tracked by this <code>BundleTracker</code> object.
<code>int</code>	<u>getTrackingCount</u> () Returns the tracking count for this <code>BundleTracker</code> object.

void	modifiedBundle (org.osgi.framework.Bundle bundle, org.osgi.framework.BundleEvent event, java.lang.Object object) Default implementation of the BundleTrackerCustomizer.modifiedBundle method.
void	open () Open this BundleTracker object and begin tracking bundles.
void	remove (org.osgi.framework.Bundle bundle) Remove a bundle from this BundleTracker object.
void	removedBundle (org.osgi.framework.Bundle bundle, org.osgi.framework.BundleEvent event, java.lang.Object object) Default implementation of the BundleTrackerCustomizer.removedBundle method.
int	size () Return the number of bundles being tracked by this BundleTracker object.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

5.2.1 context

```
protected final org.osgi.framework.BundleContext context
    Bundle context this BundleTracker object is tracking against.
```

Constructor Detail

5.2.2 BundleTracker

```
public BundleTracker(org.osgi.framework.BundleContext context,
                    int stateMask,
                    BundleTrackerCustomizer customizer)
```

Create a BundleTracker object for bundles whose state is present in the specified state mask.

Bundles whose state is present on the specified state mask will be tracked by this BundleTracker object.

Parameters:

context - BundleContext object against which the tracking is done.

stateMask - A bit mask of the ORing of the bundle states to be tracked.

customizer - The customizer object to call when bundles are added, modified, or removed in this BundleTracker object. If customizer is null, then this BundleTracker object will be used as the

BundleTrackerCustomizer object and the BundleTracker object will call the BundleTrackerCustomizer methods on itself.

See Also:

`Bundle.getState()`

Method Detail

5.2.3 addingBundle

```
public java.lang.Object addingBundle(org.osgi.framework.Bundle bundle,  
                                       org.osgi.framework.BundleEvent event)
```

Default implementation of the `BundleTrackerCustomizer.addingBundle` method.

This method is only called when this `BundleTracker` object has been constructed with a `null` `BundleTrackerCustomizer` argument. The default implementation returns the specified `Bundle` object.

This method can be overridden in a subclass to customize the object to be tracked for the bundle being added.

Specified by:

[addingBundle](#) in interface [BundleTrackerCustomizer](#)

Parameters:

`bundle` - `Bundle` being added to this `BundleTracker` object.

`event` - The bundle event which caused this customizer method to be called or `null` if there is no bundle event associated with the call to this method.

Returns:

The customized object to be tracked for the bundle added to this `BundleTracker` object.

See Also:

[BundleTrackerCustomizer](#)

5.2.4 close

```
public void close()  
    Close this BundleTracker object.
```

This method should be called when this `BundleTracker` object should end the tracking of bundles.

5.2.5 getBundles

```
public org.osgi.framework.Bundle[] getBundles()  
    Return an array of Bundle objects for all bundles being tracked by this BundleTracker object.
```

Returns:

Array of `Bundle` objects or `null` if no bundles are being tracked.

5.2.6 getObject

```
public java.lang.Object getObject(org.osgi.framework.Bundle bundle)
```

Returns the customized object for the specified `Bundle` object if the bundle is being tracked by this `BundleTracker` object.

Parameters:

`bundle` - Bundle being tracked.

Returns:

Customized object or `null` if the specified `Bundle` object is not being tracked.

5.2.7 getTrackingCount

```
public int getTrackingCount()
```

Returns the tracking count for this `BundleTracker` object. The tracking count is initialized to 0 when this `BundleTracker` object is opened. Every time a bundle is added, modified or removed from this `BundleTracker` object the tracking count is incremented.

The tracking count can be used to determine if this `BundleTracker` object has added, modified or removed a bundle by comparing a tracking count value previously collected with the current tracking count value. If the value has not changed, then no bundle has been added, modified or removed from this `BundleTracker` object since the previous tracking count was collected.

Returns:

The tracking count for this `BundleTracker` object or -1 if this `BundleTracker` object is not open.

5.2.8 modifiedBundle

```
public void modifiedBundle(org.osgi.framework.Bundle bundle,  
                           org.osgi.framework.BundleEvent event,  
                           java.lang.Object object)
```

Default implementation of the `BundleTrackerCustomizer.modifiedBundle` method.

This method is only called when this `BundleTracker` object has been constructed with a `null` `BundleTrackerCustomizer` argument. The default implementation does nothing.

Specified by:

[modifiedBundle](#) in interface [BundleTrackerCustomizer](#)

Parameters:

`bundle` - Bundle whose state has been modified.

`event` - The bundle event which caused this customizer method to be called or `null` if there is no bundle event associated with the call to this method.

`object` - The customized object for the bundle.

See Also:

[BundleTrackerCustomizer](#)

5.2.9 open

```
public void open()
```

Open this `BundleTracker` object and begin tracking bundles.

Bundle which match the state criteria specified when this `BundleTracker` object was created are now tracked by this `BundleTracker` object.

Throws:

`java.lang.IllegalStateException` - if the `BundleContext` object with which this `BundleTracker` object was created is no longer valid.

`java.lang.SecurityException` - If the caller and this class do not have the appropriate `AdminPermission[context bundle, LISTENER]`, and the Java Runtime Environment supports permissions.

5.2.10 remove

```
public void remove(org.osgi.framework.Bundle bundle)
```

Remove a bundle from this `BundleTracker` object. The specified bundle will be removed from this `BundleTracker` object. If the specified bundle was being tracked then the `BundleTrackerCustomizer.removedBundle` method will be called for that bundle.

Parameters:

`bundle` - Bundle to be removed.

5.2.11 removedBundle

```
public void removedBundle(org.osgi.framework.Bundle bundle,  
                           org.osgi.framework.BundleEvent event,  
                           java.lang.Object object)
```

Default implementation of the `BundleTrackerCustomizer.removedBundle` method.

This method is only called when this `BundleTracker` object has been constructed with a `null` `BundleTrackerCustomizer` argument. The default implementation does nothing.

Specified by:

[removedBundle](#) in interface [BundleTrackerCustomizer](#)

Parameters:

`bundle` - Bundle being removed.

`event` - The bundle event which caused this customizer method to be called or `null` if there is no bundle event associated with the call to this method.

`object` - The customized object for the bundle.

See Also:

[BundleTrackerCustomizer](#)

5.2.12 size

```
public int size()
```

Return the number of bundles being tracked by this `BundleTracker` object.

Returns:

Number of bundles being tracked.

5.3 org.osgi.util.tracker.BundleTrackerCustomizer

All Known Implementing Classes:

[BundleTracker](#)

```
public interface BundleTrackerCustomizer
```


The `BundleTrackerCustomizer` interface allows a `BundleTracker` object to customize the bundle objects that are tracked. The `BundleTrackerCustomizer` object is called when a bundle is being added to the `BundleTracker` object. The `BundleTrackerCustomizer` can then return an object for the tracked bundle. The `BundleTrackerCustomizer` object is also called when a tracked bundle has been removed from the `BundleTracker` object.

The methods in this interface may be called as the result of a `BundleEvent` being received by a `BundleTracker` object. Since `BundleEvents` are received synchronously by the `BundleTracker`, it is highly recommended that implementations of these methods do not alter bundle states while being synchronized on any object.

The `BundleTracker` class is thread-safe. It does not call a `BundleTrackerCustomizer` object while holding any locks. `BundleTrackerCustomizer` implementations must also be thread-safe.

Since:

1.4

Method Summary

<code>java.lang.Object</code>	<code>addingBundle</code> (<code>org.osgi.framework.Bundle bundle</code> , <code>org.osgi.framework.BundleEvent event</code>) A bundle is being added to the <code>BundleTracker</code> object.
<code>void</code>	<code>modifiedBundle</code> (<code>org.osgi.framework.Bundle bundle</code> , <code>org.osgi.framework.BundleEvent event</code> , <code>java.lang.Object object</code>) A bundle tracked by the <code>BundleTracker</code> object has been modified.
<code>void</code>	<code>removedBundle</code> (<code>org.osgi.framework.Bundle bundle</code> , <code>org.osgi.framework.BundleEvent event</code> , <code>java.lang.Object object</code>) A bundle tracked by the <code>BundleTracker</code> object has been removed.

Method Detail

5.3.1 `addingBundle`

```
java.lang.Object addingBundle(org.osgi.framework.Bundle bundle,  
                                org.osgi.framework.BundleEvent event)
```

A bundle is being added to the `BundleTracker` object.

This method is called before a bundle which matched the search parameters of the `BundleTracker` object is added to it. This method should return the object to be tracked for this `Bundle` object. The returned object is stored in the `BundleTracker` object and is available from the [`getBundles`](#) method.

Parameters:

`bundle` - Bundle being added to the `BundleTracker` object.

`event` - The bundle event which caused this customizer method to be called or `null` if there is no bundle event associated with the call to this method.

Returns:

The object to be tracked for the `Bundle` object or `null` if the `Bundle` object should not be tracked.

5.3.2 modifiedBundle

```
void modifiedBundle(org.osgi.framework.Bundle bundle,  
                    org.osgi.framework.BundleEvent event,  
                    java.lang.Object object)
```

A bundle tracked by the `BundleTracker` object has been modified.

This method is called when a bundle being tracked by the `BundleTracker` object has had its state modified.

Parameters:

`bundle` - Bundle whose state has been modified.

`event` - The bundle event which caused this customizer method to be called or `null` if there is no bundle event associated with the call to this method.

`object` - The tracked object for the modified bundle.

5.3.3 removedBundle

```
void removedBundle(org.osgi.framework.Bundle bundle,  
                   org.osgi.framework.BundleEvent event,  
                   java.lang.Object object)
```

A bundle tracked by the `BundleTracker` object has been removed.

This method is called after a bundle is no longer being tracked by the `BundleTracker` object.

Parameters:

`bundle` - Bundle that has been removed.

`event` - The bundle event which caused this customizer method to be called or `null` if there is no bundle event associated with the call to this method.

`object` - The tracked object for the removed bundle.

6 Considered Alternatives

6.1 Using Services to model Bundles

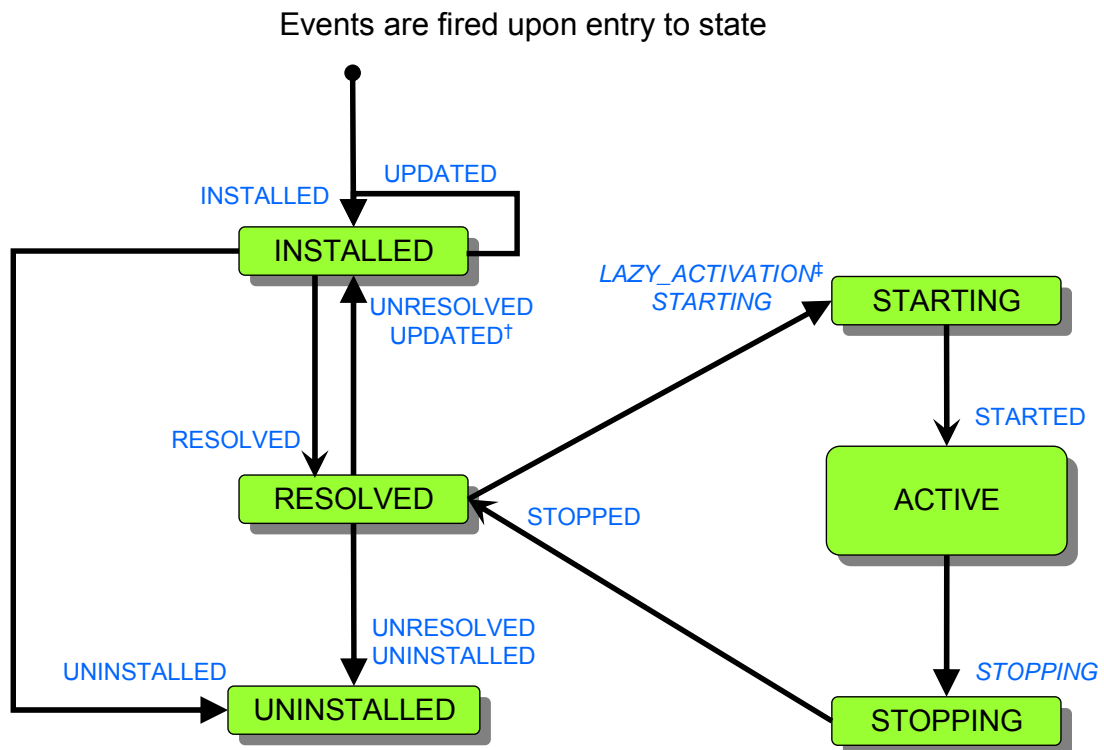
See Member Bug 501[3] for some discussion of this design alternative.

6.2 Using asynchronous Bundle Listener

The first draft of this RFC suggested allowing BundleTracker to provide the option of using either BundleListener (asynchronous) and SynchronousBundleListener. This was changed to only support SynchronousBundleListener due to issues with using asynchronous BundleListener. Following is the removed text:

The Bundle Tracker supports user configuration to use either a Synchronous Bundle Listener or the asynchronous Bundle Listener. For synchronous bundle event processing, the bundle's state is set before the event is synchronously fired and the event is delivered before the state can change again. However, if asynchronous bundle event processing is used, then the behavior of the tracker will be different since entry to some states are not visible to asynchronous bundle listeners and the time between the event firing and event delivery may prevent some bundle state transitions from being observed.

The following diagram depicts the bundle states and the event fired upon entry to those states. Note that entry to STARTING and STOPPING states is only signaled to synchronous bundle listeners and are thus not reliably observable to asynchronous bundle listeners.



† only if updated

‡ only if lazy activation; STARTING is later fired when activation commences

Events in *italics* are only delivered to synchronous bundle listeners

There are also cases when several bundle events can be fired before the first event is asynchronously delivered. During processing of the first event by the BundleTracker, the state of the bundle at the time the final event was fired is observed. This can result in the bundle becoming tracked while processing the first event. As the

remaining events are then delivered, the bundle's state does not actually change, but the Bundle Tracker will call the `modifiedBundle` method of the customizer for each additional event. There are also cases where the bundle should be removed and added back the tracker (e.g. bundle update), but delay in delivery of the STOPPED event, which might result in the bundle being removed from the tracker, until after the bundle has been restarted will result in the tracker never seeing the bundle leave the ACTIVE state and thus never being removed and then added back to the tracker.

Thus the value of supporting asynchronous bundle listeners in Bundle Tracker is dubious and we may want to consider removing it.

To deal with the above issues, two approaches are possible but still may deliver suboptimal results.

1. Process the event types in addition to the bundle state to "simulate" the bundle being removed and added if necessary. This logic could be fairly complex as it will have to map event type onto the state map. This would only be necessary for the asynchronous listener.
2. Have Bundle Tracker always use a synchronous bundle listener and wrap the `addBundleListener` call in a `doPrivileged` method to not require the caller to have the necessary permission. This would make every bundle able to synchronously be notified of bundle events which will provide a form of privilege elevation in secured systems.

7 Security Considerations

Bundle Tracker runs in the security context of the bundle using it. It doesn't provide or remove any of the security checks that are already in place for bundles.

In order to support tracking bundles synchronously, a `SynchronousBundleListener` must be used. In order to prevent elevation of privilege, the Bundle Tracker implementation must not use `doPrivilege` when registering the `SynchronousBundleListener` object. This means that the code calling the `open` method (which makes the `addBundleListener` call) and the Bundle Tracker class itself must both have the `AdminPermission[context bundle, LISTENER]` permission. In particular, the bundle containing the `org.osgi.util.tracker` package must have this permission. If the `org.osgi.util.tracker` package is delivered as part of the framework implementation, then it likely has `AllPermission` and this requirement is then met.

8 Document Support

8.1 References

- [1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
 - [2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0
 - [3]. Member Bug 501, https://www2.osgi.org/members/bugzilla/show_bug.cgi?id=501
 - [4]. Service Tracker Specification, OSGi Core Specification, R4 V4.1, Section 701
-

8.2 Author's Address

Name	BJ Hargrave
Company	IBM Corporation
Address	800 N Magnolia Av, Orlando, FL, USA
Voice	+1 386 848 1781
e-mail	hargrave@us.ibm.com

8.3 Acronyms and Abbreviations

8.4 End of Document



RFC 125 - Bundle License

Draft

9 Pages

Abstract

This RFC defines the format and rules for a Bundle-License header in the manifest

Copyright © OSGi Alliance 2008.

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	4
4 Requirements	5
5 Technical Solution	5
6 Security Considerations	7
7 Document Support	7
7.1 References	7
7.2 Author's Address	7
7.3 Acronyms and Abbreviations.....	7
7.4 End of Document.....	7

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.

Revision	Date	Comments
Initial	SEP 27 2007	Initial, Peter Kriens, aQute

1 Introduction

This RFC defines a new header for the next release. It was introduced by Bugzilla bug #483, see [3]. The original request was denied during a CPEG meeting because it required the maintenance of a licenses list, this was deemed too cumbersome for the OSGi. However, the use of a Bundle-License header was acknowledged. This small RFC designs a more acceptable form of a Bundle-License header.

2 Application Domain

It is interesting to note how many technically inclined people on software conferences tend to only discuss licensing issues. The advent of open source with its myriad of licenses, combined with the (ab)use of licenses in a very different spirit than intended has created a complex situation. The Open Source Licenses web site, see [4] has a list with 60 open source compatible licenses. Well known licenses are Mozilla Public License, Eclipse Public License, Apache Software License, and of course GPL and LGPL. These licenses differ in a grand scale sometimes, like GPL versus Apache or often quite close.

Currently, licenses are stored in artifacts in an ad hoc way, there are no rules. The requirements on how to handle the licensing of artifacts is a black art. There is very little jurisprudence in this area and there are usual lawyers involved that tend to want to err on the safe side by adding the same information many times and seem to only use upper case fonts for unknown reasons. It is unlikely that a single set of rules can match the requirements of these lawyers.

Most larger corporations have strict policies of what kind of licenses can be used and which licenses are compatible with their policy.

Most licenses are defined available over the internet via a URL. Some examples:

- <http://www.apache.org/licenses/LICENSE-2.0.txt>
- <http://www.apache.org/licenses/LICENSE-1.1>

- <http://www.gnu.org/licenses/gpl-2.0.txt>
- <http://www.gnu.org/licenses/gpl-3.0.txt>
- <http://www.gnu.org/licenses/old-licenses/lgpl-2.1.txt>
- <http://www.eclipse.org/legal/epl-v10.html>
- <http://www.opensource.org/licenses/sunpublic.php>
-

Many licenses do not have a URL, for example, for BSD only a template could be found that was embedded in a more philosophical discussion. Also, many licenses do not take versioning into account or update the URL with a new version without changing the name. For example, the URL <http://www.gnu.org/licenses/gpl.txt> refers to GPL 3 while it used to refer to GPL 2.0.

There are of course also commercial licenses. Where open source libraries have some form of organization, commercial licenses can not be assumed to have a URL. Despite the fact that many licenses have a URL, most software still include the license text in or with the artifact to avoid any confusion.

Licensing can be complicated if the artifact is covered by multiple licenses. There are two possibilities:

- Parts of the artifacts fall under different licenses
- The user has the choice of choosing between difference licenses

I do not think that you ever have the case of multiple licenses that are ALL valid, that is you always accept only one license for each resource?

In certain case, the receiver can accept different licensing rules. For example, the government often gets different conditions than ordinary citizens. Or a company could have a special deal for certain large corporations.

A very common use case that the artifact stipulates that was delivered with a license. I.e., the provider of the artifact has a single delivery artifact but stipulates the license external from the artifact.

3 Problem Description

It should possible for OSGi bundle to include licensing information about the rights of the bundle in such a way that bundles can be machine read and verified to comply with company policies.

4 Requirements

- R001 – Allow parts of the bundle to be licensed differently than other bundles
- R001 – Allow multiple licenses per part
- R002 – Allow unique identification of open source licenses
- R003 – Provide descriptive information so that a processor is not required to go online to show information to the end user.
- R004 – Allow the licenses to be stored in the bundle and connected to the applied licenses.
- R005 – Provide a human readable name that can be used to select the license information in a list
- R006 – Provide a model where licenses can be targeted at certain entities and not to be considered by others.
- R007 – Allow a provider to make it clear that the bundle's license is provided through other means. I.e. the bundle is not self descriptive regarding licenses.
- R008 – Ensure the OSGi Alliance does not have responsibility for the contents

5 Technical Solution

The following header syntax is proposed:

```
Bundle-License ::= "<<EXTERNAL>>" | ( license ( ',' license ) * )
license       ::= name ( ';' license-attr ) *
license-attr  ::= description | link | covers | local | exclusive
description   ::= 'description' '=' string
link          ::= 'link' '=' <url>
local         ::= 'local' '=' path // see ...
covers        ::= 'covers' '=' path | ''' path ( ',' path ) * '''
```

```
exclusive ::= 'exclusive' '=' bsn | "' ' bsn ( ',' bsn) * '"
```

This header has the following aspects:

- *name* – Provides a globally unique name for this license, preferably world wide but in minimum for the other clauses. Clients of this bundle can assume that licenses with the same name refer to the same license. This can for example be used to minimize the click through licenses. This name is the cardinal URL of the license, it must *not* be localized by the developer. This URL does not have to exist but must not be used for later versions of the license. It is advised to use the following structure, but this is not mandated:
 - <http://<domain-name>/licenses/<license-name>-<version>.<extension>>
- *description* – (optional) Provide the description of the license. This is a short description that is usable in a list box on a UI to select more information about the license.
- The magic name <<EXTERNAL>> is used to indicate that this artifact does not contain any license information but that licensing information is provided in some other way. This is the default contents of this header.
- *link* –(optional) Provide a URL to a page that defines or explains the license. If this link is absent, the name field is used for this purpose. This field can be localized to allow different URLs for different locales.
- *covers* – (option) Lists the paths of the bundle that are covered by this license. Paths are supposed to include the directory they point to and all sub directories. Paths should not start with a /, they are all assumed to start at the root. The default for covers is empty, indicating the whole bundle. If different licenses have overlapping values for this, then the license is assume to be a choice for the user, any of the overlapping licenses is acceptable.
- *local* – (option) A path to the license inside the bundle. The license include in the bundle has priority over the url or any license defined by the url or name. However, bundle developer must not store a license file that is not identical to the one identified by the name or url, this would be considered malice.
- *exclusive* – (option) This attribute lists the names of entities that can accept this license. In certain cases, for example the government, different licensing rules apply. If this field is set, it should be considered absent if the entity name is not recognized as the accepting party. This name is not formally defined but is assumed to be the domain name of the organization:
 - *gov*
 - *darpa.mil*
 - *ibm.com*
 - *sales.bea.com*
 - *felix.apache.org*

If multiple licenses are listed with an overlapping covers attribute then the user is free to license the resources under any of the listed licenses.

If the Bundle-License statement is absent, then this does not mean that the bundle is not licensed. Licensing could be handled outside the bundle.

A user should accept at least one license for each of the resources in the bundle to consider the bundle as accepted.

6 Proposed Specification Text

The following section is added before 3.2.1.9 Bundle-Location

6.1 Bundle-License Header

The Bundle-License header provides an optional machine readable form of license information. The purpose of this header is to automate some of the license processing required by many organizations like for example license acceptance before a bundle is used. The header is structured to provide the use of unique license naming to merge acceptance requests as well as links to human readable information about the included licenses. This header is purely informational for management agents and must not be processed by the OSGi Framework.

The syntax for this header is as follows:

```
Bundle-License ::= "<<EXTERNAL>>" | ( license ( ',' license ) * )
license        ::= name ( ';' license-attr ) *
license-attr   ::= description | link
description    ::= 'description' '=' string
link           ::= 'link' '=' <url>
```

This header has the following attributes:

- *name* – Provides a globally unique name for this license, preferably world wide but it should at least be unique with respect to the other clauses. The magic name <<EXTERNAL>> is used to indicate that this artifact does not contain any license information but that licensing information is provided in some other way. This is the default contents of this header. Clients of this bundle can assume that licenses with the same name refer to the same license. This can for example be used to minimize the click through licenses. This name should be the cardinal URL of the license, it must *not* be localized by the translator. This URL does not have to exist but must not be used for later versions of the license. It is recommended to use URLs for open source code from <http://www.opensource.org/licenses/alphabetical>. Other licenses should use the following structure, but this is not mandated:
 - <http://<domain-name>/licenses/<license-name>-<version>.<extension>>
- *description* – (optional) Provide the description of the license. This is a short description that is usable in a list box on a UI to select more information about the license.

-
- *link* –(optional) Provide a URL to a page that defines or explains the license. If this link is absent, the name field is used for this purpose. The URL is relative to the root of the bundle. I.e. it is possible to refer to a file inside the bundle.

If the Bundle-License statement is absent, then this does not mean that the bundle is not licensed. Licensing could be handled outside the bundle and the <<EXTERNAL>> form should be assumed.

Clearly, this header is informational and may not have any legal bearing. Consult a lawyer before using this header to automate licensing processing.

7 Security Considerations

The Bundle-License header has no known security implications.

8 Document Support

8.1 References

- [1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
- [2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0
- [3]. https://www2.osgi.org/members/bugzilla/show_bug.cgi?id=483
- [4]. <http://www.opensource.org/licenses>

8.2 Author's Address

Name	Peter Kriens
Company	aQute
Address	9c, Avenue St. Drézéry
Voice	+33467542167
e-mail	Peter.Kriens@aQute.biz

8.3 Acronyms and Abbreviations

8.4 End of Document



RFC 126 - Service Registry Hooks

Draft

17 Pages

Abstract

This RFC describes the means for a bundle to hook into the service registry operations.

Copyright © IBM Corporation 2008

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.....	3
0.3 Revision History.....	3
1 Introduction.....	4
2 Application Domain.....	4
3 Problem Description.....	5
4 Requirements.....	5
5 Technical Solution.....	5
5.1 Modifications to current API.....	5
5.1.1 New ServiceEvent type.....	5
5.2 New Hook Classes.....	6
5.2.1 PublishHook.....	6
5.2.2 FindHook.....	6
5.2.3 ListenerHook.....	6
5.3 Backwards Compatibility Requirements.....	6
6 Javadoc.....	7
6.1 Interface PublishHook.....	org.osgi.framework.hooks.service 7
6.1.1 event.....	7
6.2 Interface FindHook.....	org.osgi.framework.hooks.service 8
6.2.1 find.....	8
6.3 Interface ListenerHook.....	org.osgi.framework.hooks.service 9
6.3.1 initial.....	10
6.3.2 added.....	10
6.3.3 removed.....	10
6.4 Class ListenerHook.Listener.....	org.osgi.framework.hooks.service 11
6.4.1 ListenerHook.Listener.....	11
6.4.2 getBundleContext.....	12

6.4.3 getServiceListener..... 12

6.4.4 getFilter..... 12

7 Considered Alternatives.....13

7.1 Pre and post hooks.....13

7.2 Listening to specific service names.....13

7.3 Framework Proxying of Hook Generated Objects.....13

7.4 Full Manipulation Capabilities.....13

7.4.2 Exposure to Hook Generated Objects..... 14

7.4.3 AdminPermission..... 15

8 Security Considerations.....16

9 Document Support.....16

9.1 References.....16

9.2 Author's Address.....16

9.3 Acronyms and Abbreviations.....16

9.4 End of Document.....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 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.

Revision	Date	Comments
Initial	7 January 2008	Initial Draft of RFC BJ Hargrave, IBM
2 nd draft	18 May 2008	Second draft based upon CPEG feedback. Mostly small changes. Some of the methods were changed to use Collections instead of array since it is expected that data is mutated as control passes along the chain. Security documentation was added to indicate that hook require permission to manipulate the services of a bundle. BJ Hargrave, IBM

Revision	Date	Comments
3 rd draft	10 June 2008	<p>Third draft based upon CPEG feedback.</p> <p>Removed the overloaded addServiceListener method and added a new ServiceEvent type. Without the overloaded addServiceListener method, the ListenerHook class has been simplified.</p> <p>Added Exposure to Hook Generated Objects. Additional input from the EGs is needed here.</p> <p>BJ Hargrave, IBM</p>
4 th draft	10 July 2008	<p>Fourth draft based upon decisions at the CPEG f2f meeting.</p> <p>CPEG agreed, with approval of the EEG RFC 119 team, to scale this design back. The changes now do not allow the hooks to inject objects into the service registry which avoids the issue of creating new dependencies between the bundles creating and using services and the hook bundles.</p> <p>BJ Hargrave, IBM</p>

1 Introduction

This RFC details how a bundle can hook into service layer operations and influence the operation or observe the operation.

2 Application Domain

This design is targeted at bundles which need to observe and manipulate select service layer operations. In general these will be highly specialized bundles written by systems programmers. The design is not intended to be used by so-called "normal" application bundles.

3 Problem Description

The service layer operations provide no means for a bundle (external to the framework) to observe or manipulate the operations as they occur. Certain specialized bundles need to be able to alter output results of service layer find and event delivery operations to affect their purpose. Such purposes may include things like distributed service model, etc.

4 Requirements

The solution must work with the current service layer model and allow certain bundles to observe the find and listen operations and to potentially reduce the result to affect their desired goals.

The solution must be secured when java permissions are in effect.

5 Technical Solution

5.1 Modifications to current API

In addition to the API in following section, some changes to current API are needed

5.1.1 New ServiceEvent type

A new type, `MODIFIED_ENDMATCH`, is added to `ServiceEvent` to allow a listener to detect when a service property modification results in the service no longer matching the filter with which the `ServiceListener` was added. Existing `ServiceListener` implementation should properly ignore the new event type since the `ServiceEvent` class has long been documented that new types may be added. (We have added new types to the similarly designed `BundleEvent` and `FrameworkEvent` classes in prior releases.) New `ServiceListener` implementations (such as an updated `ServiceTracker`) can use the new `ServiceEvent` type to detect when a service property modification results in an end to the filter match. The new `ServiceEvent` type is only delivered to listeners which were added with a non-null filter where the filter matched the service properties prior to the modification but the filter does not match the modified service properties.

Here is some pseudo code which demonstrates how the framework should process delivery of this new event to a listener in response to the modification of service properties.

```
If (listenerFilter == null)
    /* if no filter, deliver MODIFIED event */
    deliverEvent(listener, ServiceEvent.MODIFIED);
else if (listenerFilter.match(newProperties))
    /* if filter matches new properties, deliver MODIFIED event */
    deliverEvent(listener, ServiceEvent.MODIFIED);
else if (listenerFilter.match(oldProperties))
    /* if filter does not match new properties but does
       match old properties, deliver MODIFIED_ENDMATCH event */
    deliverEvent(listener, ServiceEvent.MODIFIED_ENDMATCH);
```

ServiceTracker must be changed to use the new ServiceEvent type and always register a listener with a user supplied filter string. ServiceTracker can use MODIFIED_ENDMATCH to untrack a service and avoid having to evaluate filters in the ServiceListener implementation.

5.2 New Hook Classes

The javadoc in the next section describes the proposed new hook classes.

Service hooks are not called for service operations on other service hooks.

The following hook types are defined.

5.2.1 PublishHook

Bundles registering this service will be called during framework service publish (register, modify, and unregister service) operations. This method is called prior to service event delivery when a publishing bundle registers, modifies or unregisters a service and can filter the bundles which receive the event.

5.2.2 FindHook

Bundles registering this service will be called during framework service find (get service references) operations. This method is called during the service find operation by the finding bundle and can filter the result of the find operation.

5.2.3 ListenerHook

Bundles registering this service will be called during service listener addition and removal. The hook is notified of the collection of service listeners and what they may be listening for and well as future changes to that collection.

5.3 Backwards Compatibility Requirements

Hooks are very specialized services which are tied closely to the operations of the service layer of the framework. While every attempt to maintain the backwards compatibility of the hook api will be made, it is possible that changes or additions to the service layer API in future versions of the OSGi Core Specification make require changes to the hook API which breaks backwards compatibility.

6 Javadoc

This section contains the javadoc for the new hook classes.

Overview	Package	Class	Use Tree	Deprecated	Index	Help
			FRAMES	NO FRAMES		
PREV CLASS NEXT CLASS			<pre> if(window==top) { document.writeln('All Classes'); } </pre>			
SUMMARY: NESTED FIELD CONSTR METHOD			//--> All Classes DETAIL: FIELD CONSTR METHOD			

6.1 org.osgi.framework.hooks.service Interface PublishHook

```
public interface PublishHook
```

OSGi Framework Service Publish Hook Service.

Bundles registering this service will be called during framework service publish (register, modify, and unregister service) operations. Service hooks are not called for service operations on other service hooks.

Version:
\$Revision: 5112 \$

Method Summary	
void	event (ServiceEvent event, java.util.Collection bundles) Event hook method.

Method Detail

6.1.1 event

```
void event (ServiceEvent event,
            java.util.Collection bundles)
```

Event hook method. This method is called prior to service event delivery when a publishing bundle registers, modifies or unregisters a service and can filter the bundles which receive the event.

Parameters:

`event` - The service event to be delivered.
`bundles` - A Collection of Bundles which have listeners to which the event may be delivered. The method implementation can remove bundles from the collection to prevent the event from being delivered to those bundles. Attempting to add to the collection will result in an `UnsupportedOperationException`.

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

```

FRAMES          NO FRAMES          <!--
                if(window==top)
                {
                document.writeln('<A   HREF="../../../../allclasses-
noframe.html"><B>All                      Classes</B></A>');
                }
//--> All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

```

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

```

FRAMES          NO FRAMES          <!--
                if(window==top)
                {
                document.writeln('<A   HREF="../../../../allclasses-
noframe.html"><B>All                      Classes</B></A>');
                }
//--> All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

```

6.2 org.osgi.framework.hooks.service Interface FindHook

```
public interface FindHook
```

OSGi Framework Service Find Hook Service.

Bundles registering this service will be called during framework service find (get service references) operations. Service hooks are not called for service operations on other service hooks.

Version:

\$Revision: 5112 \$

Method Summary

<pre>void find(BundleContext context, boolean allServices, Find hook method.</pre>	<pre>java.lang.String name, java.lang.String filter, java.util.Collection references)</pre>
--	---

Method Detail

6.2.1 find

```
void find(BundleContext context,
         java.lang.Stringname,
         java.lang.Stringfilter,
         booleanallServices,
         java.util.Collectionreferences)
```

Find hook method. This method is called during the service find (for example, [BundleContext.getServiceReferences\(String, String\)](#)) operation by the finding bundle and can filter the result of the find operation.

Parameters:

- context - The bundle context of the finding bundle.
- name - The class name of the services to find or null to find all services.
- filter - The filter criteria of the services to find or null for no filter criteria.
- allServices - true if the find operation is the result of a call to [BundleContext.getAllServiceReferences\(String, String\)](#)
- references - A Collection of Service References to be returned to the finding bundle. The method implementation can remove references from the collection to prevent the references from being returned to the finding bundle. Attempting to add to the collection will result in an UnsupportedOperationException.

Overview	Package	Class	Use Tree	Deprecated	Index	Help
			FRAMES	NO	FRAMES	<!--
			if(window==top)		{	
PREV CLASS NEXT CLASS			document.writeln('<A HREF="../../../../allclasses-		Classes);	
			noframe.html">All		}	
			/--> All Classes			
SUMMARY: NESTED FIELD CONSTR METHOD			DETAIL: FIELD CONSTR METHOD			

Overview	Package	Class	Use Tree	Deprecated	Index	Help
			FRAMES	NO	FRAMES	<!--
			if(window==top)		{	
PREV CLASS NEXT CLASS			document.writeln('<A HREF="../../../../allclasses-		Classes);	
			noframe.html">All		}	
			/--> All Classes			
SUMMARY: NESTED FIELD CONSTR METHOD			DETAIL: FIELD CONSTR METHOD			

6.3 org.osgi.framework.hooks.service Interface ListenerHook

```
public interface ListenerHook
```

OSGi Framework Service Listener Hook Service.

Bundles registering this service will be called during service listener addition and removal. Service hooks are not called for service operations on other service hooks.

Version:

\$Revision: 5115 \$

Nested Class Summary

static class	ListenerHook.Listener A Service Listener wrapper.
--------------	--

Method Summary

void	added (ListenerHook.Listener listener) Add listener hook method.
void	initial (ListenerHook.Listener [] listeners) Initial listeners hook method.
void	removed (ListenerHook.Listener listener) Remove listener hook method.

Method Detail

6.3.1 initial

```
void initial(ListenerHook.Listener[] listeners)
```

Initial listeners hook method. This method is called when the hook is first registered to provide the hook with the set of service listeners which were had been added prior to the hook being registered. This method is only called once. However, due to the timing of other bundles adding or removing service listeners, calls to [added\(org.osgi.framework.hooks.service.ListenerHook.Listener\)](#) or [removed\(org.osgi.framework.hooks.service.ListenerHook.Listener\)](#) may occur before the call to this method.

Parameters:

`listeners` - An array of listeners which are listening to service events.

6.3.2 added

```
void added(ListenerHook.Listener listener)
```

Add listener hook method. This method is called during service listener addition. This method will be called once for each service listener added after this hook had been registered.

Parameters:

`listener` - A listener which is now listening to service events.

6.3.3 removed

```
void removed(ListenerHook.Listener listener)
```

Remove listener hook method. This method is called during service listener removal. This method will be called once for each service listener removed after this hook had been registered.

Parameters:

`listener` - A listener which is no longer listening to service events.

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

```

FRAMES          NO FRAMES          <!--
                if(window==top)
                {
                document.writeln('<A   HREF="../../../../allclasses-
noframe.html"><B>All                      Classes</B></A>');
                }
//--> All Classes

```

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

```

FRAMES          NO FRAMES          <!--
                if(window==top)
                {
                document.writeln('<A   HREF="../../../../allclasses-
noframe.html"><B>All                      Classes</B></A>');
                }
//--> All Classes

```

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

6.4 org.osgi.framework.hooks.service Class ListenerHook.Listener

```
java.lang.Object
└─org.osgi.framework.hooks.service.ListenerHook.Listener
```

Enclosing interface:

[ListenerHook](#)

```
public static class ListenerHook.Listener extends java.lang.Object
```

A Service Listener wrapper. This immutable class encapsulates a [ServiceListener](#) and the bundle which added it and the filter with which it was added. Objects of this type are created by the framework and passed to the [ListenerHook](#).

Constructor Summary

[ListenerHook.Listener](#) ([BundleContext](#) context, [ServiceListener](#) listener, java.lang.String filter)
Create a Service Listener wrapper.

Method Summary

BundleContext	getBundleContext () Return the context of the bundle which added the listener.
java.lang.String	getFilter () Return the filter with which the listener was added.
ServiceListener	getServiceListener () Return the service listener.

Methods inherited from class java.lang.Object

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

Constructor Detail

6.4.1 ListenerHook.Listener

```
public ListenerHook.Listener(BundleContext context,  
                             ServiceListener listener,  
                             java.lang.String filter)
```

Create a Service Listener wrapper.

Parameters:

`context` - The context of the bundle which added the listener.
`listener` - The ServiceListener object.
`filter` - The filter with which the listener was added.

Method Detail

6.4.2 getBundleContext

```
public BundleContext getBundleContext ()
```

Return the context of the bundle which added the listener.

Returns:

The context of the bundle which added the listener.

6.4.3 getServiceListener

```
public ServiceListener getServiceListener()
```

Return the service listener.

Returns:

The service listener.

6.4.4 getFilter

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

Return the filter with which the listener was added.

Returns:

The filter with which the listener was added. This may be `null` if the listener was added without a filter.

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

```
FRAMES      NO FRAMES      <!--
              if(window==top)
              document.writeln('<A   HREF="../../../../allclasses-
noframe.html"><B>All           Classes</B></A>');
              }
//--> All Classes
```

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: NESTED | FIELD | [CONSTR](#) | [METHOD](#)

DETAIL: FIELD | [CONSTR](#) | [METHOD](#)

7 Considered Alternatives

7.1 Pre and post hooks

The original proposal was based upon pre and post hook methods being called before and after each service operation. The issue with that proposal was matching the pre and post hook calls to a specific service operation (since service operations can be nested). The current proposal is a stack-based mechanism as each hook is responsible for calling to the next step of the operation and in general provides a simpler model.

7.2 Listening to specific service names

Adding a new `addServiceListener` method was rejected in favor of adding a new `ServiceEvent` type to notify `ServiceListeners` added with a filter string that a property modification occurred which causes the service to no longer match the filter.

The `BundleContext.addServiceListener` method must be overloaded with a new method which takes an array of service names:

```
addServiceListener(ServiceListener listener, String[] names)
```

This new method is an alternate to the filter variant of `addServiceListener` which takes a specific list of service names to watch. The strings in the `names` parameter allow the use of the "*" wildcard like filters. Using this new method is the equivalent of calling `addServiceListener` with the filter string:

```
"(|(objectClass="+name[0]" ... |(objectClass="+name[n]" )"
```

This new any new function, but provide a means of registering interest in specific service names which can allow the framework to make certain optimizations, rather than trying to look inside a potentially complicated filter string.

7.3 Framework Proxying of Hook Generated Objects

An alternative to Exposure to Hook Generated Objects was rejected. This alternative would require the framework to proxy wrap all hook generated objects such that no other bundle (including other hooks) would be exposed to hook generated objects. Then, when a hook service is unregistered, for all object generated by the hook, the framework generated proxy wrapper would dereference the hook generated objects and route around the dereferenced object.

While this is simpler and less perturbing than the requirements in Exposure to Hook Generated Objects, this alternative raised many correctness issues. For example, a hook may have modified the properties of a service during registration. Now that the hook is removed, the service property on the currently registered service should be removed. Doing this is harder for the framework to detect and "unwind".

Due to these correctness issues, this alternative was rejected.

7.4 Full Manipulation Capabilities

The goals of this RFC have been scaled back to avoid the issues with exposure of hook generated objects to bundle and the associated dependencies which are created and their cleanup.

7.4.1.1 Chained Hook Classes

Service hooks are not called for service operations on other service hooks. Each hook type (except for `ListenerHook` which is not chained) is chained and called in a sequence during the processing of a service operation. Each hook in the chain must call the chain object to continue processing the chain. The "pre" phase of a hook are the operations that occur prior to calling the chain object to pass control to the next hook in the chain. This typically includes modifying parameters before calling the chain object. The "post" phase of the hook are the operations that occur after the chain object returns. This may include modifying the return value of the chain object.

In general, the hooks are very powerful mechanism that can be used to perform many interesting functions. But because the hooks are powerful, care must be avoid misusing or abusing them.

The following hook types are defined.

7.4.1.2 *PublishHook*

Bundles registering this service will be called during framework service publish (register service) operations. This hook allows one to influence the service publish (registration) operation.

Since a *PublishHook* implementation may create and return a *ServiceRegistration* object (which wraps the original, framework created *ServiceRegistration* object), care must be taken by the framework during automatic service unregistration at bundle stop. Since the hook created *ServiceRegistration* object may manage resources that should be freed when the service is unregistered, during automatic service unregistration at bundle stop, the framework **MUST** call the *unregister* method on the *ServiceRegistration* object returned to the bundle which registered the service to allow any hook defined processing to occur.

7.4.1.3 *FindHook*

Bundles registering this service will be called during framework service find (get service references) operations. This hooks allows one to influence the service find operation. This hook supports things such as just-in-time service publication by examining the find parameters during the pre phase processing of the hook.

7.4.1.4 *BindHook*

Bundles registering this service will be called during framework service bind (get service object) operations. This hook allows one to influence the service bind operation. This hook supports thing such as proxy wrapping of the service object during the post phase processing of the hook.

7.4.1.5 *EventHook*

Bundles registering this service will be called during framework service event delivery. This hook allows one to influence the set of bundles which receive a service event.

7.4.2 Exposure to Hook Generated Objects

During the course of service operations, a bundle may become exposed to hook generated objects. For example, a hook could create and return a *ServiceRegistration* wrapper or create and return a service object wrapper to allow the hook to insert behavior onto the object's operations.

But hooks are services registered from ordinary bundles. Those bundles can be stopped and their hook services unregistered or the hook bundle may decide for some reason to unregister the hook service. This can leave the hook generated objects dangling in the system. In order to properly manage these dependencies from the client bundle to the hook service, the framework must track these dependencies and properly resolve them.

When a hook service returns a hook generated object, if the service operation completes normally resulting in the client bundle becoming dependent on the hook generated object, the framework must mark the client bundle dependent upon the hook service. If the client bundle is stopped, then all dependencies on the hook service are removed. However, if the hook service is unregistered, the client bundles that are dependent on that hook service must all be stopped and restarted so that they will re-execute the service operations without the presence of the removed hook service.

Given potential perturbation issues with multiple bundles being restarted multiple times of a set of hook bundles having a set of hooks service are updated, some care must be taken to avoid this. However, there are several scenarios to consider:

- Simple unregistration of hook service – The hook bundle may decide to unregister the hook service for some reason. The unregistration is not as a result of the hook bundle stopping and it is unknown when or if the hook

service will be reregistered. In this case, there are no other event to wait for to decide to handle the client bundle dependencies on the hook service.

- Hook bundle is stopped – The hook bundle is stopped for some reason (that is not part of an update operation). As a result, all hook services from the hook bundle are unregistered. A client bundle may have dependencies on multiple hook services from the hook bundle. We don't want to restart the client bundle multiple times (once for each unregistered hook service that the client bundle is dependent upon). Also, it is unknown when or if the hook bundle will be restarted and its hook service reregistered. In this case, there are no other event to wait for to decide to handle the client bundle dependencies on the hook services.

- One or more hook bundles are updated – A set of hook bundles are updated (hopefully all stopped, all updated and all restarted). It is also possible a `refreshPackages` call may be made on these hook bundles prior to restarting them. If we know the hook services are being unregistered as part of a larger update operation, we will want to coordinate restarting the dependent client bundles. In fact, we would want to stop the dependent client bundles before we stop the hook bundles and restart them after we restart the hook bundles. But this would require the framework to know things that it does not know. It has been suggested to delay stopping and restarting the dependent client bundles until `refreshPackages` is called. This does leave the dependent client bundles operating while the hook bundles have been stopped and updated.

From a correctness point of view, restarting the dependent client bundles must be done when the hook service is unregistered. However, there is the potential for multiple restarts. Additional input from CPEG and EEG is requested here.

7.4.3 AdminPermission

The proposed new action to be added to AdminPermission is rejected. Since the capability of the hook is reduced, such fine grained control is not necessary.

```
AdminPermission.SERVICE_HOOK
```

This new type allows control over whether the hook can manipulate the service operations of the target bundle.

8 Security Considerations

When Java permissions are in effect, this design is secured by ServicePermissions.

The bundle registering the various hook services must have the necessary ServicePermission.REGISTER. Since there are various hook services, we have fine grained control over what specific hooks a bundle can register.

9 Document Support

9.1 References

- [1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
- [2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

9.2 Author's Address

Name	BJ Hargrave
Company	IBM Corporation
Address	800 N Magnolia Av Orlando, FL
Voice	+1 386 848 1781
e-mail	hargrave@us.ibm.com

9.3 Acronyms and Abbreviations

9.4 End of Document



Accessing Exit Values from Applications

Draft

9 Pages

Abstract

This RFC describes an approach for accessing the exit value from an application launched using an Application Descriptor in Application Admin.

Copyright © IBM 2008

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.....	3
0.3 Revision History.....	3
1 Introduction.....	3
2 Application Domain.....	3
2.1 Terminology.....	3
3 Problem Description.....	4
4 Requirements.....	4
4.1 Use Cases.....	4
4.1.1 Launching Native Commands.....	4
4.1.2 Eclipse RCP Applications.....	4
5 Technical Solution.....	5
5.1 Method <code>getExitValue</code>	5
5.1.1 <code>ApplicationException</code> JavaDoc.....	5
5.1.2 <code>ApplicationHandle</code> JavaDoc.....	5
6 Considered Alternatives.....	6
6.1 No Block and timeout.....	6
7 Security Considerations.....	7
7.1 Exit Value Sensitivity.....	7
7.2 Exit Value Memory Leak.....	8
8 Document Support.....	8
8.1 References.....	8
8.2 Author's Address.....	8
8.3 Acronyms and Abbreviations.....	8
8.4 End of Document.....	8

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.

Revision	Date	Comments
Initial	Oct 22 2007	Initial draft using information provided by bug 433 Thomas Watson, IBM, tjwatson@us.ibm.com
	Oct 26 2007	Added timeout parameter to getExitValue Thomas Watson
	Aug 05 2008	Final clean up, removed question about where the service property should be and the security questions

1 Introduction

The Application Admin specification is used to manage an environment with many different types of applications that are simultaneously available. In some application environments when an application is finished the application is allowed to return an exit value. Currently the Application Admin Specification does not allow access to an exit value returned by an application when it is finished.

This RFC describes an approach for accessing an exit value when an application is finished.

2 Application Domain

2.1 Terminology

Exit Value – The result returned by an application after it has finished. This value is application and application container specific

3 Problem Description

Many application environments allow applications to return an exit value. For example, when launching a process on many operating systems the process will return an integer value indicating a success or failure of the execution. This is evident in the `java.lang.Process#exitValue()` method.

Other application environments may allow more complicated exit values for returning results from the application. For example, in the Eclipse RCP application container applications are allowed to return arbitrary Objects.

4 Requirements

1. Clients which launch an application must be able to access the exit value of an application once it has finished.
 2. An application must be able to return an arbitrary value for the exit value once it has finished execution.
 3. Clients must be able to determine when an application exit value is available
 4. Existing applications and application containers must continue to work without modification
-

4.1 Use Cases

4.1.1 Launching Native Commands

In Java, native commands can be executed by calling `Runtime#exec(...)`. This returns a `Process` object which can be used to track the execution of the command and retrieve exit values. This is similar to how an `ApplicationHandle` can be used to track an application instance in Application Admin. Some scenarios may want to register native commands as `ApplicationDescriptor` services. The `ApplicationHandle` implementation could then use the `Process#waitFor` method to implement the lifecycle of the `ApplicationHandle`. But there is no way for the client to get access to the `Process#exitValue` from an `ApplicationHandle`.

4.1.2 Eclipse RCP Applications

In the Eclipse Rich Client Platform an application container is defined for running applications defined by bundles installed in the framework. An RCP application bundle uses the extension registry to specify an application definition declaratively with an extension (a la `plugin.xml`). The application definition specifies an entry point to the application which is used to run the application. The entry point is a class from the application bundle which implements an interface specific to the RCP application container. This class is used by the application container to launch the application and retrieve the exit value from the application when it is finished. Applications in the RCP application container are allowed to return any object type they like as exit values.

5 Technical Solution

5.1 Method `getExitValue`

A new method is added to the `ApplicationHandle` class to get the exit value from an application instance. This method allows a timeout value to be specified. If a timeout is specified this method will block until either the application has terminated or the expiration of the timeout. If the application has terminated then the exit value is returned otherwise an `ApplicationException` is thrown.

The `getExitValue` method will not be abstract and the default implementation will throw an `UnsupportedOperationException`. This is necessary to allow existing applications and application container implementations to continue to work without modification or re-compilation. A new `ApplicationHandle` service property is added to indicate if the application instance supports exit values (`application.supports.exitvalue`).

Key Name	Type	Default	Description
<code>application.supports.exitvalue</code>	Boolean	FALSE	Specifies whether the application instance supports an exit value.

5.1.1 ApplicationException JavaDoc

5.1.1.1 APPLICATION_EXITVALUE_NOT_AVAILABLE

The exit value is not available for an application instance because the instance has not terminated.

5.1.2 ApplicationHandle JavaDoc

5.1.2.1 APPLICATION_SUPPORTS_EXITVALUE

String org.osgi.service.application.ApplicationHandle.APPLICATION_SUPPORTS_EXITVALUE = "application.supports.exitvalue"

The property key for the supports exit value property of this application instance.

5.1.2.2 GetExitValue

Object org.osgi.service.application.ApplicationHandle.getExitValue(long timeout)

Returns the exit value for the application instance. The timeout specifies how the method behaves when the application has not terminated. A negative, zero or positive value may be used.

- negative - The method does not wait for termination. If the application has not terminated then an `ApplicationException` is thrown
- zero - The method waits until the application has terminated
- positive - The method waits until the application has terminated or the timeout has expired. If the timeout has expired and the application has not terminated then an `ApplicationException` is thrown.

The default implementation throws an `UnsupportedOperationException`. The application model should override this method if exit values are supported.

Parameters:

timeout The maximum time in milliseconds to wait for the application to timeout.

Returns:

the exit value for the application instance. The value is application specific.

Throws:

`UnsupportedOperationException` if the application model does not support exit values.

`InterruptedException` if the wait has been interrupted.

`ApplicationException`

6 Considered Alternatives

6.1 No Block and timeout

The original proposal did not have the ability to block and wait for an application instance to terminate. If clients wanted to wait for an application to terminate they could use standard OSGi ServiceEvents to do so. But this left a rather difficult programming pattern for simple scenarios that simply wanted to launch an application and wait for the exit value. Something like the following (likely buggy code) would have to be used:

```
ApplicationHandle handle = app.launch(null);
ServiceReference[] handleRefs =
    context.getServiceReferences(
        ApplicationHandle.class.getName(),
        "(service.pid=\"" + handle.getInstanceId() + "\")");
if (handleRefs != null) {
    final boolean[] unregistered = new boolean[] {false};
    ServiceTrackerCustomizer handleCustomizedTracker =
        new ServiceTrackerCustomizer() {
            public Object addingService(ServiceReference reference) {
                return reference;
            }
            public void modifiedService(ServiceReference reference, Object service)
            {
            }
            public void removedService(ServiceReference reference, Object service)
            {
                synchronized (unregistered) {
                    unregistered[0] = true;
                    unregistered.notifyAll();
                }
            }
        };
    ServiceTracker appTracker =
        new ServiceTracker(context, handleRefs[0], handleCustomizedTracker);
    appTracker.open();
    synchronized (unregistered) {
        while (!unregistered[0]) {
            unregistered.wait();
        }
    }
    appTracker.close();
}
Object exitData = handle.getExitValue();
```

It was determined that the best thing to do would be to allow for a timeout value instead which would make the above code look like this:

```
ApplicationHandle handle = app.launch(null);
```

```
Object exitData = handle.getExitValue(0);
```

7 Security Considerations

7.1 Exit Value Sensitivity

The exit value could contain sensitive data. Clients must have `ServicePermission` to acquire the `ApplicationDescriptor` to launch the application instance or to acquire the `ApplicationHandle` service.

7.2 Exit Value Memory Leak

The `ApplicationHandle` must maintain a reference to the application value for the lifetime of the `ApplicationHandle` object (i.e. until it is GC'ed). `ApplicationHandle` objects cannot throw away the result as soon as they are unregistered. If `ApplicationHandles` are prevented from GC'ing and the exit value consumes large amounts of memory/resources then a drastic memory leak will occur.

8 Document Support

8.1 References

- [1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
- [2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

8.2 Author's Address

Name	Thomas Watson
Company	IBM
Address	
Voice	(512) 838 4533
e-mail	tjwatson@us.ibm.com

8.3 Acronyms and Abbreviations

8.4 End of Document



RFC 0129 Initial Provisioning Update

Draft

6 Pages

Abstract

Initiali Provisioning is gaining in popularity but has at least one problem: the extra field. This RFC describes a remedies to problems with IP.

Copyright © OSGi Alliance 2008

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.....	3
1 Introduction	3
2 Application Domain	3
3 Problem Description	4
4 Requirements	4
5 Technical Solution	4
5.1 Manifest header.....	4
5.2 Extension.....	5
5.3 Priority.....	5
5.4 Bundle MIME Type.....	5
6 Security Considerations	5
7 Document Support	6
7.1 References.....	6
7.2 Author's Address.....	6
7.3 End of Document.....	6

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.

Revision	Date	Comments
Initial	NOV 24 2007	Peter Kriens, aQute, Initial draft
2 nd draft	4 Feb 2008	Updated based upon CPEG discussions 2008-01-31. BJ Hargrave, IBM

1 Introduction

This RFC investigates any issues with the Initial Provisioning specification and proposes remedies.

2 Application Domain

Initial Provisioning specifies the process how an initially blank system Service Platform can be configured from a single URL. The process is described in the compendium in The OSGi Service Platform Specification.

For this RFC, the use of the extra field is the key issue. The extra field in the ZIP entry is used to indicate the type of the entry. A MIME type is used for this. The following types are supported:

- text `text/plain;charset=utf-8`
- binary `application/octet-stream`
- bundle `application/x-osi-bundle`
- bundle-url `text/x-osi-bundle-url`

3 Problem Description

The R4 IP specification requires that the MIME type of a Provisioning Dictionary is stored in the extra field of a Zip entry. Though the extra field is well supported in the Java util.jar package classes, it is not well supported in build tools. This makes the creating of the Provisioning Dictionary extra hard.

4 Requirements

- Provide an alternative to the IP use of the extra field that is supported by standard tools

5 Technical Solution

5.1 Manifest header

The list of entries to be processed can be specified by this new manifest header.

```
InitialProvisioning-Entries ::= entry ( ',' entry )*
```

```
entry ::= path ( ';' parameter )*
```

The entry is the path name of a resource in the JAR file. The following attributes is recognized:

- mime – Describes the mime type of the entry. This must be one of the four valid mime types of the ZIP entry.
- type – Is one of the 4 types: text, binary, bundle, or bundle-url

If neither the mime or type parameter entry is specified for an entry, then the type will be inferred from the extension of the entry.

5.2 Extension

The type of an entry can be inferred from its extension. The following extensions to type mappings are proposed:

Type	Extension	Description
text	.txt	The IP Dictionary must contain a String. The file must be encoded in UTF-8
binary	Not .txt, .url, .jar	Any file not recognized as one of the other three types is treated as binary
bundle	.jar	A jar file is treated as a bundle to be installed
bundle-url	.url	A text file, UTF-8 encoded, containing a URL to a bundle that will be installed

If the new manifest header is not specified and the extra field of the entry is not specified, then an entry whose extension matches one of these extensions will be processed as an IP entry.

5.3 Priority

An implementation of the Initial Provisioning must determine the mime type of an entry in the following priority order:

1. The extra field - This is necessary to prevent existing systems from failing because an extension does not match the intent.
2. The InitialProvisining-Entries manifest header
3. The extension of the entry.

5.4 Bundle MIME Type

The current mime type used in IP is application/x-osgi-bundle. Subsequent to the creation of the IP specification, OSGi registered an official mime type for bundles: application/vnd.osgi.bundle. The original application/x-osgi-bundle mime type may continue to be used as an alternative to the new application/vnd.osgi.bundle type. However, applications must continue to recognize the older type.

6 Security Considerations

There are no additional security considerations for these modifications.

7 Document Support

7.1 References

- [1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
- [2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0
- [3]. OSGi Bundle Mime Type,
<http://www.iana.org/assignments/media-types/application/vnd.osgi.bundle>

7.2 Author's Address

Name	Peter Kriens
Company	Aqute
Address	9c, Avenue St. Drezery
Voice	+33 467542167
e-mail	Peter.Kriens@aQute.biz

7.3 End of Document



RFC 0132 Command Line Interface and Launching

Draft

41 Pages

Abstract

This RFC describes a proposed specification for a Command processing interface for the OSGi Framework.

Copyright © OSGi Alliance 2008

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.....	4
0.3 Revision History.....	4
1 Introduction.....	5
2 Problem Description.....	5
2.1 Framework Launching.....	5
2.2 Command Interface.....	5
3 Requirements.....	6
3.1 Non Functional.....	6
3.2 Launching.....	6
3.3 Command Names.....	6
3.4 Shell.....	7
3.5 Shell Commands.....	7
3.6 Source Providers.....	7
4 Technical Solution.....	8
4.1 Launching.....	8
4.2 Life Cycle Issues.....	9
4.3 Properties.....	9
4.4 Starting Procedure.....	10
4.5 Shell Design.....	11
4.6 Command Service.....	13
4.7 Thread IO Service.....	13
4.8 Shell syntax: TSL.....	14
4.8.1 Introduction to TSL (Tiny Shell Language).....	14
4.8.2 Program Syntax.....	15
4.8.3 Examples of Syntax usage.....	17
4.9 Standard IO Handling.....	17
4.10 Command Provider Discovery.....	17
4.11 Other Commands.....	19
4.12 Piping.....	19
4.13 Command Calling.....	19

4.13.1 Remove Variable.....	20
4.13.2 Assignment.....	20
4.13.3 Single Value.....	20
4.13.4 Call Cmd.....	20
4.13.5 Message Send	20
4.14 Argument Coercion.....	21
4.15 Converters.....	22
4.16 Printing or Not.....	23
4.17 TSL In OSGi.....	23
4.18 Services and their Commands.....	24
4.19 Help.....	25
5 Javadoc.....	25
5.1	org.osgi.framework.launch
Interface SystemBundle.....	25
5.1.1 SECURITY.....	26
5.1.2 STORAGE.....	27
5.1.3 LIBRARIES.....	27
5.1.4 EXECPERMISSION.....	27
5.1.5 ROOT_CERTIFICATES.....	27
5.1.6 WINDOWSYSTEM.....	27
5.1.7 init.....	28
5.1.8 waitForStop.....	28
5.2	org.osgi.service.command
Interface CommandProcessor.....	29
5.2.1 COMMAND_SCOPE.....	29
5.2.2 COMMAND_FUNCTION.....	30
5.2.3 createSession.....	30
5.3	org.osgi.service.command
Interface CommandSession.....	30
5.3.1 execute.....	31
5.3.2 execute.....	32
5.3.3 close.....	32
5.3.4 getKeyboard.....	32
5.3.5 getConsole.....	33
5.3.6 get.....	33
5.3.7 put.....	33
5.3.8 format.....	33
5.3.9 convert.....	34
5.4	org.osgi.service.command
Interface Converter.....	34
5.4.1 CONVERTER_CLASSES.....	35
5.4.2 INSPECT.....	35
5.4.3 LINE.....	35
5.4.4 PART.....	36
5.4.5 convert.....	36
5.4.6 format.....	36
5.5	org.osgi.service.command
Interface Function.....	37
5.5.1 execute.....	37
5.6	org.osgi.service.threadio
Interface ThreadIO.....	38

5.6.1 setStreams..... 39

5.6.2 close..... 39

6 Alternatives.....39

 6.1 Considered setParentBundle.....39

7 Security Considerations.....40

8 Document Support.....40

 8.1 References.....40

 8.2 Author’s Address.....40

 8.3 Acronyms and Abbreviations.....41

 8.4 End of Document.....41

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.

Revision	Date	Comments
Initial	05 MAR 2008	Peter.Kriens@aQute.biz
Additional text	03 APR 2008	Added a large number of sections, mainly booting and more details about the processing of scripts. Also completely changed the API and added a problem description and requirements section. Changes are so massive that it was not that useful to track changes (and I forgot anyway)
Update	15 MAY 2008	More details in booting, minor update in the language aspects
Update	31 JUL 2008	Changed some method names, updated Javadoc and added to build
Update	5 AUG 2008	Updated javadoc section. Changed “boot” to “launch”.

1 Introduction

This RFC is an solution for RFP 99 Command Provider and RFP 80 Framework booting. This RFC outlines the interfaces necessary to implement command line shells in OSG frameworks as well as providing a generic start up solution for different implementations of frameworks.

2 Problem Description

This RFC addresses the problem of standardized external access. The purpose of this RFC is to allow any OSGi framework to be launched, configured, and controlled externally from a console, telnet session, serial port or script. The framework must enable a set of basic commands that are supported by all implementations, but it must enable that bundles can provide additional commands.

2.1 Framework Launching

Every OSGi framework must invent its own technology for getting started. Therefore, code that may need to launch the framework from various places must write code that is proprietary for the particular OSGi vendors' implementation, if they wish to support more than one framework.

In enterprises, where the issues of vendor lock-in can cause a barrier to adoption of the system, this issue becomes magnified. While it should certainly be the case that each vendor can supply add-ons and extra features, the standard portions of all OSGi frameworks could be encapsulated in a Framework Launching specification. Having such a specification would increase the consistency and quality of compliant OSGi frameworks, and allay fears about vendor lock-in.

2.2 Command Interface

There is a need for a service that allows human users as well as programs to interact with an OSGi based system with a line based interface: a shell. This shell should allow interactive and string based programmatic access to the core features of the framework as well as provide access to functionality that resides in bundles.

Shells can be used from many different sources it is therefore necessary to have a flexible scheme that allows bundles to provide shells based on telnet, the Java 6 Console class, plain Java console, serial ports, files, etc.

Supporting commands from bundles should be made very lightweight and simple as to promote supporting the shell in any bundle. Commands need access to the input and output streams. Commands sometimes need to store state between invocations in the same session.

There is a need for a very basic shell functionality in small embedded devices, however, the design should permit complex shells that support background processing, piping, full command line editing, and scripting. It is possible that a single framework holds multiple shells.

The shell must provide a means to authenticate the user and the commands must be able to investigate the current user and its authorizations, preferably through standardized security mechanisms. To minimize footprint, it must also be possible to implement a shell without security.

3 Requirements

3.1 Non Functional

- Lightweight to allow shells for low footprint devices
- Allow shells with piping, background, scripting, etc.
- Make commands trivial to implement
- Make it easy to connect the shell to different sources.
- Provide an optional security framework based on existing security facilities
- Minimize the cost of a command (e.g. do not require eager loads of objects implementing commands)
- Support use of existing code by making a design that closely follows practices for command line applications in Java.

3.2 Launching

- The solution should allow for scripting languages to behave similarly for all compliant OSGi frameworks.
- The solution must allow for starting, restarting and stopping compliant OSGi frameworks without prior knowledge of the framework.
- The solution should allow for starting compliant frameworks in the same or different processes (though it would be OK to fall back on the Java Process verbs if necessary)
- The solution should handle the problems associated with launching multiple OSGi frameworks inside the same JVM process space.
- The solution must be able to handle vendor specific extensions.

3.3 Command Names

- Provide a list of basic command signatures to manage the framework so they are consistent among implementations.

3.4 Shell

- Provide interface to execute a string as command
- Allow other bundles to implement commands
- Allow other bundles to provide a connection to: telnet, console, serial port, etc.
- Provide help for each command
- Provide a means to disambiguate commands with the same name
- Provide a means to disambiguate when there are multiple shells
- Authenticate the user
- Provide programmatic access to the shell, that is, a program generates the commands.

3.5 Shell Commands

- Read input from user or previous command
- Write output to user or next command
- Allow sessions, i.e. group commands over a period of time, allowing them to share state
- Provide usage information of the command
- Allow commands to be protected with permissions
- Provide access to the authenticated user via User Admin (though User Admin may not be present)
- Optional: Allow computer readable meta information about the commands to support forms
- Optional: Provide formatting rules + library to standardize look and feel for output. This could consist of routines to show tables in a consistent form.
- Commands should not have to do low level parsing of command line arguments.
- Commands should be able to have access to the command line arguments
- Commands must be able to get access to the console input
- Commands must be able to use the keyboard input stream

3.6 Source Providers

- Provide an easy way to allow bundles to connect the shell to sources like telnet, serial ports, etc.

4 Technical Solution

4.1 Launching

The solution for the launch process is quite straightforward. Frameworks must designate a class that has an empty constructor and implements the following interfaces

- `org.osgi.framework.Bundle`
- `org.osgi.framework.launch.SystemBundle` (which extends `Bundle`)

The current specification provides a very detailed description of the System Bundle. The instantiated object represents this system bundle in an unstarted state (`RESOLVED|INSTALLED`). However, the system bundle must be able to provide a valid Bundle Context that can be used to install applications. These applications must not be started before the system bundle itself is started. The `SystemBundle` provides a number of methods that can be used to configure the framework **before** starting it. The System Bundle object can be configured, started, stopped, reconfigured, and started again, ad nauseum.

The System Bundle must implement the following methods.

- `init(Properties)` - The `Properties` object (which may be null is optionally backed by other `Properties` such as the System properties) must be used for the framework properties. The framework must use this properties as the only source by using `getProperty` (not `get`) so that the configurator can use the `Properties` linked behavior. If the properties object is null, useful defaults should be used to make the framework run appropriately in the current VM. I.e. the system packages for the current execution environment should be properly exported. Any persistent area should be defined in the current directory with a framework implementation specific name.

The `init` method should be called before the `start` method is called. If the `start` method is called before the `init` method has been called, then `start` must call the `init` method with a null parameter. After the `init` method has been called, the system bundle must have a valid bundle context.

A system bundle can be reinitialized by first stopping it and then calling `init(Properties)` again. Calling `init` when the system bundle is started must throw an `Illegal State Exception`.

This method is not thread safe and must only be called from a single thread.

- `waitForStop()` - This method waits until the bundle is stopped and completely finished with the cleanup. This method will wait until someone calls, or has called, `stop`. If the system is not `ACTIVE`, then this method returns directly. This method is thread safe and can be called by different threads.

The following is a short example of creating a Felix framework, adding bundles to it, launching the framework and then waiting for it to stop. This assumes that some bundle (for example a shell) will initiate the stop command to the system bundle.

- `f = new org.apache.felix.Felix`
- `f init`
- `context = f bundleContext`
- `$context installBundle http://www.aQute.biz/repo/aQute/sample.jar`

- f start
- f waitForStop

This example makes the main thread wait for the framework to finish.

4.2 Life Cycle Issues

A framework must never do `System.exit(n)` when stopped. It is the responsibility of the configurator to exit.

A framework can be make repeated start/stop cycles. The semantics of stopping a framework are described in the core specification. It is not possible to change the properties. The options in the start and stop methods are ignored.

what happens with update?

4.3 Properties

The configurator can set the following properties. In addition, it can also add framework implementation dependent properties.

org.osgi.framework.version	set by framework implementation
org.osgi.framework.vendor	set by framework implementation
org.osgi.framework.language	set by configurator, but framework should provide a default when not set
org.osgi.framework.executionenvironment	set by configurator, but framework should provide a default when not set
org.osgi.framework.processor	set by configurator, but framework should provide a default when not set
org.osgi.framework.os.version	set by configurator, but framework should provide a default when not set
org.osgi.framework.os.name	set by configurator, but framework should provide a default when not set
org.osgi.supports.framework.extension	set by framework
org.osgi.supports.bootclasspath.extension	set by framework
org.osgi.supports.framework.fragment	set by framework to true
org.osgi.supports.framework.requirebundle	set by framework to true
org.osgi.framework.bootdelegation	set by configurator, framework provides empty default.
org.osgi.framework.system.packages	set by configurator, but framework should provide a default when not set
org.osgi.framework.security	The name of a Security Manager class with public empty constructor. A valid value is also true, this means that the framework should instantiate its own security manager. If not set, security could be defined by a parent framework or there is no security. This can be detected by looking if there is a security manager set

	### ???
org.osgi.framework.storage	A valid file path in the file system to a directory that exists. The framework is free to use this directory as it sees fit. This area can not be shared with anything else. If this property is not set, the framework should use a file area from the parent bundle. If it is not embedded, it must use a reasonable platform default.
org.osgi.framework.libraries	A list of paths (separated by path separator) that point to additional directories to search for platform specific libraries
org.osgi.framework.command.execpermission	The command to give a file executable permission. This is necessary in some environments for running shared libraries.
org.osgi.framework.root.certificates	Points to a directory with certificates. ###??? Keystore? Certificate format?
org.osgi.framework.windowssystem	Set by the configurator but the framework should provide a reasonable default.

4.4 Starting Procedure

This RFC does not propose a proper shell script for the launch procedure. The Bundle object approach to framework creation is sufficient to allow almost any Java compatible script language to be used. An example of such a script language is the shell as proposed in this document (tsl), but any script language will work: Jython, Jruby, Beanshell, Bex, Groovy, etc.

For example in Groovy

```

framework = "org.apache.felix.framework.Felix"
lib = "file:jar/felix.jar"

///// Generic
storage = new File("osgi- storage").getAbsolutePath()
storage.mkdirs()

properties = [
'org.osgi.framework.system.packages':"org.osgi.framework, \
    org.osgi.service.packageadmin, \
    org.osgi.service.startlevel, \
    javax.sql",
'org.osgi.framework.storage' : storage.absolutePath,
'org.osgi.service.http.port' : '8080'
]

this.class.classLoader.rootLoader.addURL( new URL(lib) )

clazz = Class.forName(framework)
systemBundle = constructor.newInstance()
systemBundle.init(properties)

ctx      = systemBundle.bundleContext
servlet = ctx.installBundle("http://www.osgi.org/repository/servlet.jar")
osgi     = ctx.installBundle("http://www.osgi.org/repository/osgi.jar")
webrpc   = ctx.installBundle("http://www.aqute.biz/uploads/Code/aQute.webrpc.jar")
suduko   = ctx.installBundle("http://www.aqute.biz/uploads/Code/aQute.sudoku.jar")

```



```
http =
ctx.installBundle("http://www.knopflerfish.org/repo/jars/http/http_all-2.1.0.jar")

http.start()
webrpc.start()
suduko.start()

systemBundle.start()
```

The shell language, outlined in the following sections, can also be used in the same way. However, this is combined in a launcher program that gets the framework library and system bundle class from the command line paramaters:

```
org.osgi.framework.system.packages =`
    org.osgi.framework,
    org.osgi.service.packageadmin,
    org.osgi.service.startlevel,
    javax.sql`
org.osgi.framework.storage = $user.home/osgi
org.osgi.service.http.port = 8080

start
ctx = bundleContext
addCommand ctx $ctx
servlet = installBundle http://www.osgi.org/repository/servlet.jar
osgi = installBundle http://www.osgi.org/repository/osgi.jar
webrpc = installBundle http://www.aqute.biz/uploads/Code/aQute.webrpc.jar
suduko = installBundle http://www.aqute.biz/uploads/Code/aQute.sudoku.jar
http = installBundle
http://www.knopflerfish.org/repo/jars/http/http\_all-2.1.0.jar
```

This design has the tremendous advantage that each organization can use its own script language. Due to the abstraction of the system bundle, it is trivial to create a launcher that can be combined with any compliant framework. It is therefore not deemed wise to standardize the script syntax for the launch process, there are already a sufficient number around.

However, it is advantageous to take advantage of the command provider interface. This model is described later, but it is based on services. Using the script language approach as defined here, it is quite easy to search the service registry for new commands. The groovy meta model or the undefined command catch functions can make this quite transparent.

4.5 Shell Design

The drivers of this design have been:

- Core Engine Implementable in < 30k
- Very easy to add new commands
- Leverage existing mechanisms

The basic idea of the design is that there are three parts. The bundle that interacts directly with the user. This bundle handles the IO streams and parses out one or more lines of text, called the “program”. This program creates a Command Session from the selected Command service. This IO processor then gets a command from

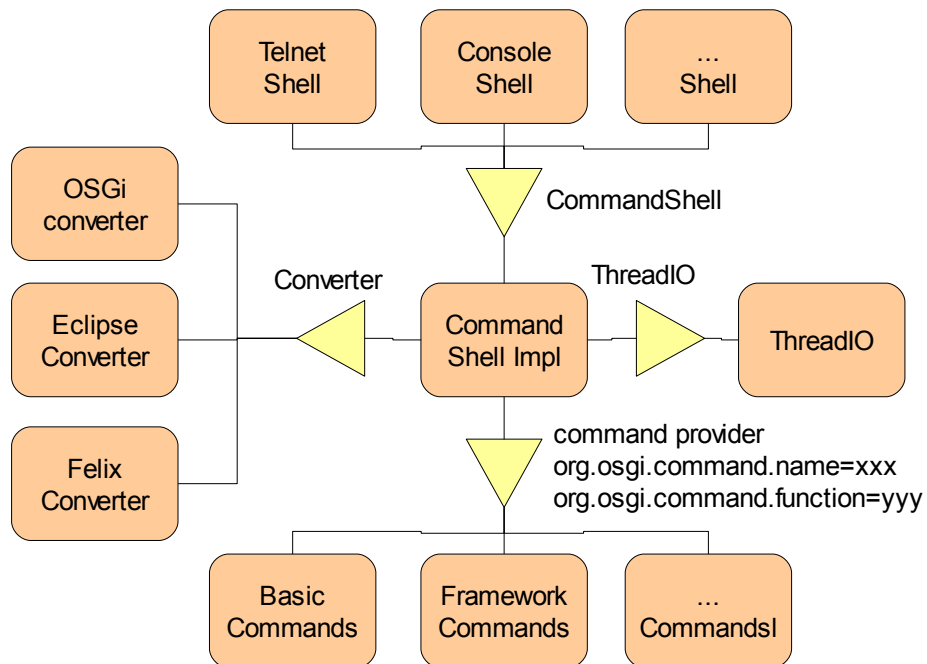
the input and gives it to a command session execute method. The command session parses the program, and executes it. The session then returns an Object result.

The command is executed synchronously. The shell will execute all *commands* in the program. These commands are implemented by services. A service can be registered with list of `COMMAND_FUNCTION` properties. These properties list the commands (potentially wildcarded) that a service can provide to the shell. These functions do not require a specific prototype, the shell matches the parameters to the function using parameter *coercion*. The type information available in the reflection API is used to convert the strings in the input to specific types.

Each command can print to `System.out` and it can retrieve information from the user (or previous command in the pipe) with `System.in`. Each command can also return an object. It is possible to retrieve the last result in the pipe through the future.

Therefore, the Command Shell service consists of three distinct parts:

- Command Processor service - This service is used by bundles that can connect the shell to an outside interface like: Telnet, Console, Web, SSH, etc. These bundles get a Command Shell service and use this service to execute their commands.
- Command Provider service - Command implementations can register this service to provide commands. Commands are methods on the service. The names (and help) of the methods can be listed through properties. There is no actual Command Provider interface because a service property allows any service to provide commands.
- Converter Service – Provide facilities to convert from strings to specific types and from specific types to strings.
- ThreadIO Service - Commands can use `System.in`, `System.out`, and `System.err` to interact with the user. However, this requires that different commands are separated in their output. This RFC therefore defines a service (likely a Framework service) that can multiplex the System IO streams.



4.6 Command Service

The command service consists of the following interfaces

- **CommandProcessor** – The engine is running the scripts. It has no UI of its own. A UI (telnet, web, console, etc, is expected to create a session from this engine. The Command Processor service is registered by the implementer of the script processor.
- **CommandSession** – The command session represents the link between a UI processor (telnet) and the command processor. It maintains a set of variables that can be set by the UI processor as well as from the script. Commands should maintain any state in the session. The session is also associated with a keyboard stream as well as a console stream. This allows commands to directly talk to the user, regardless if they are piped or not.
- **Converter** – A converter service is registered with a list of classes at the `osgi.converter.classes` properties. A converter can convert an object to a `CharSequence` object and it can convert an object of an arbitrary class (though likely a `CharSequence`) to an instance of one of the listed classes. Converters are heavily used to minimize the command code. The Command Processor will attempt to coerce parameters and results in the required instances using converters.
- **Function** – A function is an executable piece of code. Commands providers can add Function objects to their arguments and execute them. This allows commands that implement iteration blocks, if statements, etc.

The following code executes a small program, assuming it is injected with a `CommandProcessor` called `cp`:

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
CommandSession session = cp.createSession(System.in,bout,System.out);
Object result = session.execute("bundles|grep aQuote");
String s = new String(bout.toByteArray());
```

4.7 Thread IO Service

The Thread IO service is a framework service that guards the singletons of `System.out`, `System.in`, and `System.err`. The interface is quite simple, it consists of two methods:

- **setStream(InputStream,PrintStream,PrintStream)** – Associate the given streams with the current thread. Any output on the current thread using any of the System Print Streams will in effect be redirected to the appropriate system stream. Input will come from the given input stream. This method can be repeated multiple times for a thread. That is, an implementation must stack the streams per thread. Streams may be null, in that case they refer to the last set stream or the default if no streams are set.
- **close()** - Cancel (or pop) the streams from the thread local stack. If no more streams are available, use the value of the original System streams.

Usage of the Thread IO service is very straightforward but care must be taken that exceptions do not leave streams on the stack. For example, the following code grabs the output:

```
String grab(ThreadIO threadio ) {
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    PrintStream pout = new PrintStream(out);
    threadio.setStreams(null,pout,pout);
    try {
        System.out.println("Starting ...");
```

```
doWhatever();
System.out.println("... Done");
} catch(Throwable t) {
    t.printStackTrace();
}
finally {
    threadio.close();
}
pout.flush();
return new String( out.toByteArray() );
}
```

Additional issues:

- Implementations of Thread IO must use weak references to the stream objects and no longer use them when they hold the only reference.
- If a framework is embedded then the threadio service must attempt to reuse the threadio implementation of the parent framework.

4.8 Shell syntax: TSL

The syntax of the shell should be simple to implement because the framework must provide a parser for this syntax. On the other hand, a more powerful syntax simplifies the implementation of the commands. For example, when Microsoft introduced a command line shell, it did not support piping. As a consequence, each command had to implement functionality to page the output. There are other examples like handling of variables, executing subcommands, etc.

The shell syntax must also be easy to use by a user. That is, a minimum number of parentheses, semicolons, etc. Some compatibility with the Unix shells like bash is desired so for users to not have to learn completely new concepts. Then again, the current popular shells have a convoluted syntax because they added more and more features over time.

An OSGi shell syntax can rely 100% on the fact that there is a Java VM. As shown in the launching section, this makes it easy to control the framework and implement a shell with Java. However, a shell implies that the users directly type the commands as they go. The requirements for a shell are therefore different than the requirements for a programming language. However, in contrast with a shell like bash that must be totally text based, it seems a waste not to tie the shell language closely to the Java object model. Though there are many script languages, there seems to be no shell language for Java that provide such a syntax. Jacl comes close but has the disadvantage that it brings its own function library derived from tcl. Beanshell comes close from the other side but has a syntax that is virtually the same as Java, which contains a lot of cruft characters. A new syntax can reuse the concepts of tcl but tie the language close to the Java language. I.e., no need for separate function libraries.

4.8.1 Introduction to TSL (Tiny Shell Language).

The tsl language consists of a tokenizer that converts a command line to strings and then interprets those strings as method calls on Java objects where the arguments are converted to the requested type. Some syntactic sugar is added to minimize typing. The language supports variables, which are real objects. Do not confuse tsl with a completely string based approach. The type information available in the Java VM is used to infer types and convert to proper objects as much as possible.

Some examples:

```
$ echo Hello World
```

```
Hello World
```

The first token is the command name because it is a simple string, and is a *command*. A command represents both an *instance* and the *method* to be called on that instance. This is in first instance confusing. However, the way it works is quite simple. Objects implementing a command (that is, having a method with a command name) are registered as a variable with a structure. The same object can be registered under many different names. That is, if an object implements `ls` and `cd`, then it is registered as `file:ls`, and `file:cd`.

As can be seen in this example, the actual name of the command is a structured name consisting of a `<scope>` and a function name, separated by a `:'`. I.e. in the earlier example the command `*:echo` is searched because the scope is not defined.

The commands are coming from the variable space of the session. It must be possible to register commands by creating variables of the proper name.

A command is represented by a Function object. In the hello world examples, the execute method is called on this object with two CharSequence parameters: `["Hello","World"]`. Because the number of arguments of echo is variable, it is declared with an array of Object.

```
public CharSequence echo( Object[] args ) {
    StringBuilder sb = new StringBuilder();
    for ( Object arg : args )
        sb.append(arg);
    return sb;
}
```

Methods can print to System.out, but are normally expected to return an object. Returning an object allows the result to be used in other commands. Tsl will print out the object to standard out if it is not used as a value in another command, for example with piping. If a program contains multiple statements, only the last value is printed out. Converter services are used to print out the objects in a proper format.

4.8.2 Program Syntax

The basic structure of a *program* is a set of statements separated by a vertical bar.

```
program ::= statements ( '|' statements ) *
```

The *statements* are executed in parallel, whereby the output of the earlier statement is the input of the next statements. These statements are executed in separate threads. Statements consist of a sequence of statement, separated by a semicolon.

```
statements ::= statement ( ';' statement ) *
```

A statement is initially a set of tokens. This means that they must be tokenized in preparation for a command execution.

```
statement ::= ( WS* token )+
```

A token is parsed out of the input. Important is the first character, this determines the type of the token. If the first character is a Java Identifier part (digit, alpha, underscore, etc), then the token is parsed until an unescaped WS is found or one of the special characters is found. with the following rules:

```
token ::= JIP (JIP | ^SPECIAL )*
        | ^SPECIAL+
        | '<' <recursive> '>'
        | '{' <recursive> '}'
        | '[' <recursive> ']'
        | '(' <recursive> ')'
        | ''' [^] '''
```

```

| '"' [^"] '"'
| '$' token
WS      ::= <Character.isWhiteSpace>
JIP     ::= <Character.isJavaIdentifierPart - $>
SPECIAL ::= [=|;<{[$,]
OPERATORS ::= [!~`#$%^&*~:~/?@.]

```

The <recursive> indicates that the token should be parsed until a matching closing bracket is found, the parsing should take escaping and strings into account finding the match. I.e. <<<<>>>, matches, as does <'><'>, as does {}}}}}}}}}. Though the content is expected to be a valid program, the tokenizer does not have to verify this, nor is this mandated. Implementing such a tokenizer is quite straightforward and requires very little code. This is the reason that the syntax does not use the program clause recursively but instead comments that it is a recursive match.

Character handling. Some characters can be escaped from special processing with the backslash character. These characters will be interpreted as their normal value without any special meaning. Some other characters escaped will give special codes:

```

\\          Backslash (2dh###)
\t         Tab (whitespace) 09h)
\b         backspace (08h)
\f         Form Feed (0ch)
\n         New Line (0ah)
\r         Return (0dh)
\u9999    Unicode
<lf | cr> Make newline a space
\         Escaped whitespace

```

need some further work on escaping ...

Whitespace is defined by Java. Unless escaped or in a string, whitespace has no meaning.

The tokens that are parsed out of the input have a special meaning depending on their first character:

- < - The less than sign indicates a direct execution block. Everything between this sign and the closing greater than sign must be a valid program. This is equivalent to a function call or the `..` operator in a unix shell. This operator was considered but unfortunately there is no difference between the opening and closing in unix, requiring complex escaping with recursive use. Everything between the opening and '<' and matching '>' belongs to the direct execution. That is, the direct execution can contain be recursive to any reasonable depth.
- '{' - The closure character. The closure contains statements. It is defined until the matching closing brace '}'. Closures are not interpreted by the shell but passed as is a Closure object the command. The first character will be the {. Closures can be nested to any reasonable depth. No unescaping is done. This is the responsibility of the receiver. The purpose of this method is to allow commands to use closures. If a command declares a Function argument, then tsl must convert a closure to a Function. Closures can refer to any parameters by using the variables \$it (the first argument), \$args (an array of all the arguments) and \$0-\$9, argument at position 0 through 9.
- '[' - The array character. An array is a whitespace separated sequence of *entry*. This is stored in a Collection or Map and can be coerced to arrays, and collection types. Arrays/Maps can be recursively defined.
- '"' and "'" - The quote characters define a string. Everything between the quote and its matching end quote is passed as a String. All characters will be unescaped when passed as an argument.

- '\$' - The variable character. If the next character is a '{' then the code between this and the matching '}' is a value, which can be a full program again. The value of this program in runtime is used to lookup a value in the System properties.
- '(' - The expression character. Everything between this character and the closing ')' defines a filter expression. This token can be coerced into a filter or a string for a function. A filter can use '(' and ')' recursively.

An array is a sequence of *token* separated by whitespace.

```
array      ::= (WS* entry)*
entry     ::= token ( WS* '=' token )*
```

4.8.3 Examples of Syntax usage

In the following examples it is assumed that there is an echo command which prints the output to System.out.

4.9 Standard IO Handling

The original shells in Apache Felix, Knopflerfish, and Equinox used special handling of input and output (if input was supported). A huge disadvantage of this method is that it requires all the command to live in a special context; making the commands hard to test. Reuse of existing code is also harder because it is likely that any io must be adapted.

This RFC therefore proposes the use standard Java input output with the System.in, System.out, and System.err streams. This means that any Java program using these streams will work. However, these streams are singletons and Java does not provide a general way to share these singletons between bundles. This would create conflicts if multiple shells were running on the system. Even if one shell runs this would be problematic because a pipeline uses different threads that need different IO streams.

This requires a service that can multiplex the IO streams based on the current thread. In practice this is almost trivial to do (a test class uses less than 50 lines, where most are actually whitespace) with Thread Local Variables. However, the key problem is to the synchronization between the different users because it requires replacing the existing System.in, out, and err with a special multiplexing class. Just like the URL services, these are singletons. We therefore need a (framework?) service that allows bundles to associate IO streams with a thread. This is a very useful function in itself.

The setStreams method will associate the given IO streams with the current thread. Any code using System.out, System.in, or System.err will use the given streams instead of the standard streams. The close method will restore the previous configuration (the streams will be pushed on a stack). If the bundle that used this service is stopped, then the stack of streams will be removed.

The Shell service uses this mechanism to associate the streams from the shell drivers with the commands, as well as for the piping.

4.10 Command Provider Discovery

Command Provider discovery is based on the OSGi service model. Any service can be used as a command provider.

Dedicated command providers must register their service with two properties:

- `osgi.command.scope` - This property defines the name of the command provider. This name is not normally used because the function names are unique. However, if the function names are no longer unique, then this scope can be used to disambiguate.
- `osgi.command.function` - The name of the function. This is a simple or an array property, so many names can be listed. This function name should match to a public method in the service object.

For example, the following code is a DS that provides a few utility commands:

```
public class Tools {
    public void grep(String match) throws IOException {
        Pattern p = Pattern.compile(match);
        BufferedReader rdr = new BufferedReader(
            new InputStreamReader(System.in));
        String s = rdr.readLine();
        while (s != null) {
            if (p.matcher(s).find()) {
                System.out.println(s);
            }
            s = rdr.readLine();
        }
    }
    public void echo( Object[] args) {
        StringBuffer sb = new StringBuffer();
        for ( Object arg : args )
            sb.append(arg);
    }
}
```

This command provider can support two commands: `echo` and `grep`. The DS scheme for this command would look like:

```
<component name="com.acme.Tools">
  <implementation class="com.acme.Tools"/>
  <property name="command.scope" value="acme.tools"/>
  <property name="command.function" value="
    grep
    echo"/>
  <service>
    <provide interface="com.acme.Tools"/>
  </service>
</component>
```

The properties provide sufficient information for the Command Shell to find the providers. Note that it is not necessary register the service as a Command Provider, the properties suffice. This makes it possible to register these properties on an existing service. For example, the Configuration Admin could just register the following properties:

```
osgi.command.scope = 'cm'
osgi.command.function = { 'createFactoryConfiguration',
  'getConfiguration', 'listConfigurations' }
```

This will enable shell scripts like:

```
cfg = configuration com.acme.pid
$cfg update [port=23 host=www.acme.com]
```

Or, for the Log Service


```
command.scope = 'log'  
command.function = 'log'
```

```
log 2 "hello world"
```

4.11 Other Commands

Any other commands can be added to the shell by storing them in the session variables. Command names are scoped like `<scope>:<function>`. The value of this variable can be a plain object, or it can be an instance of Function. If it is an instance of Function, it can be directly executed. Else, the method with the function name is called upon it.

For example, the following code registers a function for each public method:

```
void addCommand(CommandSession session, String scope, Object target ) {  
    Method methods[] = target.getClass().getDeclaredMethods();  
    for ( Method m : methods ) {  
        if ( Modifiers.isPublic(m.getModifiers()))  
            session.put( scope + ":" + m.getName(), target );  
    }  
}
```

Tsl also has *closures*. Closures implement the Function interface. The follow code will add a command written in tsl:

```
$ my:echo = { echo xx $args xx }  
Closure ...  
$ my:echo Hello World  
xxHelloWorldxx
```

4.12 Piping

Piping seems to introduce a significant complexity in the command processing. However, it turns out that it can be implemented with very little code that easily outweighs the advantages if the increased simplicity of the commands. The key example is of course the “less” or “more” command. Many of the other commands can generate output that is too much to fit the screen. Using piping, the output can be paged through a centralized command. Functions like `grep`, `uniq`, etc. are all impossible without piping.

Normal unix shells have io redirection. It was chosen to not implement this, but instead use commands. There are two commands that can redirect io:

```
cat <file>+          concatenate files and pipe to output  
tac [-f <file>]     receive input and store it in file or return object
```

That is, to get the output of bundles as a string in a variable:

```
output = <bundles | tac>
```

4.13 Command Calling

When the program is tokenized it basically consists of set of statements. A statement consists of a set of tokens. These tokens are parsed into objects. I.e. a reference to a variable is looked up, a `<>` is executed recursively, a `{}` is translated to a Closure, an array into a List or Map. The next step depends on the list of values.

```
<string> '='          remove variable <string>  
<string> '=' <value> + execute values as statements, set result as variable  
<string>             if cmd exists, call <string>, else no such command
```

```
<string> <value>+          call cmd <string> with arguments
<object> <value> <value>*  send message <value> to <object> with remaining args
```

4.13.1 Remove Variable

The form `<name> '='` is used to remove a variable from the local scope. There must be no token behind the '=', not even an empty token, like for example "

```
java.lang.vm =
```

4.13.2 Assignment

An assignment has the form:

```
<property name> '=' statement
```

The statements part must be executed as if it was entered on an empty line, the result of this execution is stored in the variable. For example:

```
jre-1.6 = javax.xml.parsers, javax...
org.osgi.framework.systempackages =  ${jre-${java.specification.version}}
```

4.13.3 Single Value

This is an interesting case. If the single value is a string, then it can be a command name, or an object that needs to be returned. If it is not a `CharSequence`, it is assumed to be an object and it is the value of the statement. If it is a `CharSequence`, a command with the given name is searched. If it is found, it is executed according to the normal rules. If not, it is assumed that the string value suffices as result.

```
abcefg          // result no such command
$shell         // returns the content of the variable
<abcef>        // returns no such command
```

4.13.4 Call Cmd

When the first argument is a string, it is assumed to be a command name. This command is looked up in the list of available Command Providers. If this is not found, an error is raised unless there are no parameters. In that case the name is returned as the result.

If a Command Provider is found, then this object is the target for a message send, the name of the message being the command name. See discovery. The arguments are matched as discussed in the "Argument Coercion" section.

```
echo 1
```

4.13.5 Message Send

If the first argument is not a string but the second is, then a message send is assumed. The method with the given name is matched to the remaining arguments.

```
shell = install http://www.acme.com/b/shell.jar // sets a bundle object
$shell start // starts the bundle
```

If there are no arguments and no method matches, then the name should be treated as a property name. The method name should be adjusted to the beans get property design pattern (i.e. `xyz` becomes `getXYZ`). If this method also does not exist, a field with the given name should be tried. I.e. the following command should return the requested level as an integer:

```
log LOG_WARNING
```

4.14 Argument Coercion

In the end, a statement consists of an assignment or the call of a Java function. When a Java function is called, it is necessary to match the arguments to the correct method. This requires that arguments are coerced in their correct type.

First finding the proper command in the service. If the cmd is a reserved word in java (static, final, new, etc.) then the command name must be prefixed with a '_' because otherwise the command could not be implemented in Java. That is, if the command is "new", the method name should be "_new".

The comparison with the method name must be done case insensitive. I.e. installBundle is the same as installbundle.

The bean syntax must be supported. That is, a command like bundles must find the method getBundles and setBundles and isBundles. See the beans design patterns for proper converting a name to a getter.

Then, the method must be found. The shell should fetch all the declared methods of the service and try to match the given parameters to all the methods with the command name. This requires matching the arguments given in the program to the arguments of the method. Matching of the methods is done with the following priority:

1. The first declared method where all arguments can be properly coerced
2. If too few arguments are specified, pad with null the method with the maximum matching arguments. Done
3. If too many arguments are specified, the find the first method which has an array at the end and where the remaining arguments can be coerced into to the array type. Done.
4. An instance "main(Object[])" method
5. The static main(String[]) method

If none is found, a NoSuchMethodException must be thrown.

If the first type of a command is a CommandSession, then the Command Shell must insert the current session in this parameter. This is a way for a command to receive the shell session itself. This can be used to recursively execute commands, to get access to the keyboard or console stream, or to get and put variables. The keyboard stream is for example necessary to do a more command.

Arguments are no strings, they are proper objects. Variables can refer to objects, arrays are objects, and also the result of a direct command (<>) can result in a proper object. Matching these objects to a method is non-trivial. In the following section r is the receiving type (defined in the method) and g is the given type. The priority for coercion is:

1. r is assignable from g, use g
2. r is an array and g is a Collection, convert g to an array r and coerce its members recursively
3. Iterate over all converters that can handle the requested type as listed in the properties, until one returns a non-null value. Order is service.ranking and the service.id.
4. r is not primitive and has String constructor, convert g to string and use constructor to convert g to r.

5. `r` is a primitive and `g` is the matching class, convert `g` to primitive value and use it
6. fail

4.15 Converters

In most shells, the formatting of the input is the majority of work. The `tsl` has attempted to minimize this work with the Converter services.

A Converter service registers itself with a list of class names that it can convert or format. `Tsl` uses these services to print returned objects or to coerce arguments for method calls.

The service property is `osgi.converter.classes`. Its value is a single string or an array of strings, reflecting the classes this converter can convert or print. For conversion, inheritance is not taken into account. For printing, `tsl` must start with the implementation class, then its superclass, recursively. If no match is found, it should try to find all implemented interfaces in the class and its ancestors. For each converter it should combine the output.

hmmm, not sure I like this.

When `tsl` needs to convert an object to a class or print an object of a specific class, it will call the registered Converter objects in the following order:

- filtered by matching class
- sorted by `service.ranking`, `service.id`

A Converter service implements 2 methods:

- `Object convert(Class, Object)` – Convert object to the given class. Return null if this can not be done.
- `CharSequence format(Object, int)` – Convert an object to a Char Sequence using the `int` parameter as a hint. This hint can be `INSPECT`, `LINE`, or `PART`. For an `INSPECT`, the output can be a multiline columnar output of any reasonable level. A `LINE` format must make the object look good in a table when different objects of the same type are printed below each other. It is allowed to use multiple line outputs as long as the format works well in a table. A `PART` format is used to identify the object. E.g. a name or identifier. The `PART` format should be usable in the `convert` method when a `CharSequence` is the object to be converted. `INSPECT`, `LINE`, and `PART` are ordered. That is, when printing an `INSPECT`, the next level should be to format an object with `LINE`, etc.

The following code shows a simple converter for Bundles that only recognizes the bundle id. (A real one should also look for symbolic name and version, or for the location, or maybe even a filter). The printing should be much better in aligning columns.

```
import org.osgi.framework.*;
import org.osgi.service.command.*;

public class BundleConverter implements Converter {
    BundleContext context;

    BundleConverter(BundleContext context) {
        this.context = context;
    }

    public Object convert(Class type, Object source) {
        if (type != Bundle.class)
```

```
        return null;

    if (source instanceof Number) {
        source = source.toString();
    }

    if (source instanceof CharSequence) {
        long id = Long.parseLong(source.toString());
        return context.getBundle(id);
    }
    return null;
}

public CharSequence format(Object o, int level, Converter escape) {
    if (!(o instanceof Bundle))
        return null;

    Bundle b = (Bundle) o;
    StringBuffer sb = new StringBuffer();
    switch (level) {
    case INSPECT:
        cols(sb, "Symbolic Name", b.getSymbolicName());
        cols(sb, "Version", b.getHeaders().get("Bundle-Version"));
        cols(sb, "State", b.getState());
        cols(sb, "Registered Services", escape.format(b
            .getRegisteredServices(), level + 1, escape));
        // ...
        break;

    case PART:
        sb.append(b.getSymbolicName()).append(";").append(
            b.getHeaders().get("Bundle-Version"));
        break;

    case LINE:
        sb.append(" ").append(b.getState()).append(" ").append(
            b.getLocation());
        break;
    }
    return sb;
}

void cols(StringBuffer sb, String label, Object value) {
    sb.append(label);
    for (int i = label.length(); i < 24; i++)
        sb.append(' ');
    sb.append(value).append('\n');
}
}
```

4.16 Printing or Not

In principle, `tsl` must only print the object when it would otherwise get lost. It will therefore only print the object when a command is piped because in that case there is nobody to use the resulting object. In all other cases, the object is kept.

4.17 TSL In OSGi

If the use of `tsl` is OSGi related then it will have registered commands for all the public methods on the `BundleContext`, `StartLevelService`, `PackageAdmin`, and `PermissionAdmin` (if present). The actual bundle context

in use is from the IO Processor. The scope of the bundle context commands must be osgi. The actual Bundle Context must be from the Bundle that got the Command Processor. This is usually the IO processor, e.g., the telnet or console program.

Additionally, the following services should be supported:

- *more* – Page the input
- *grep <regex>* – Search in the input and only transfer to output the input that matches the <regex>.
- *each <iterable> <closure>* - Iterate over the iterable and call the closure for each element. The first argument (\$it) is the iterable element.
- *echo <value> ** - Print the value to the System.out without any spaces in between.
- *quit* – Quits the shell
- *exit* – Exits the framework

This makes any public method available:

```
$ bundles | grep aQute
0003 ACT biz.aQute.bnd file:/Ws/aQute/aQute.bnd/bnd.jar
0023 ACT biz.aQute.fileinstall file:/Ws/aQute/aQute.fileinstall/fileinstall.jar
```

The bundles command is found as osgi:bundles. This command has the current Bundle Context (defined by the env again) as the implicit receiver and the name bundles as function. There is no function bundles on Bundle Context, but in the spirit of the beans, tsl must also look at no arg get methods. I.e. the method name is getBundles but for brevity, bundles must be matched as well. This method returns null or a Bundle[].

Because this command is input from the user, and piped, tsl will print it to the pipe. The grep function looks like:

```
public CharSequence grep(String match) throws IOException {
    Pattern p = Pattern.compile(match);
    BufferedReader rdr = new BufferedReader(
        new InputStreamReader(System.in));
    List<String> list = new ArrayList<String>();
    StringBuilder sb = new StringBuilder();
    String s = rdr.readLine();
    while (s != null) {
        if (p.matcher(s).find()) {
            list.add(s);
            s = rdr.readLine();
        }
    }
    return list;
}
```

4.18 Services and their Commands

Implementations of services are recommended to provide commands for their service, this is quite straightforward and described in a later section. For example, assume that the implementation of the Configuration Admin has registered its method as commands. I.e. all its public methods are available.

```
$ my.pid = configuration my.pid; $my.pid update [port=5012 host=www.aQute.biz ]
```

The configuration command is executed against the Configuration Admin service. This returns a Configuration object. In this case, it is stored in the my.pid variable. In the next statement, we call the update method on the Configuration object and set a dictionary.

Ok, one more example, based on the fact that the Configuration Admin is available as commands:

```
$ listConfigurations (service.pid=com.acme.*) | grep port
```

4.19 Help

tbd

5 Javadoc

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.1 org.osgi.framework.launch Interface SystemBundle

All Superinterfaces:
[Bundle](#)

```
public interface SystemBundle extends Bundle
```

This interface should be implemented by framework implementations when their main object is created. It allows a configurator to set the properties and launch the framework. TODO The javadoc in this class need a good scrub before release.

Version:

\$Revision: 5214 \$

Field Summary

static java.lang.String	EXECPERMISSION
-------------------------	--------------------------------

The command to give a file executable permission.

static java.lang.String	LIBRARIES A list of paths (separated by path separator) that point to additional directories to search for platform specific libraries
static java.lang.String	ROOT_CERTIFICATES Points to a directory with certificates.
static java.lang.String	SECURITY The name of a Security Manager class with public empty constructor.
static java.lang.String	STORAGE A valid file path in the file system to a directory that exists.
static java.lang.String	WINDOWSYSYSTEM Set by the configurator but the framework should provide a reasonable default.

Fields inherited from interface org.osgi.framework.Bundle

[ACTIVE](#), [INSTALLED](#), [RESOLVED](#), [START_ACTIVATION_POLICY](#), [START_TRANSIENT](#), [STARTING](#), [STOP_TRANSIENT](#), [STOPPING](#), [UNINSTALLED](#)

Method Summary

void	init (java.util.Properties configuration) Configure this framework with the given properties.
void	waitForStop (long timeout) Wait until the framework is completely finished.

Methods inherited from interface org.osgi.framework.Bundle

[findEntries](#), [getBundleContext](#), [getBundleId](#), [getEntry](#), [getEntryPaths](#), [getHeaders](#), [getHeaders](#), [getLastModified](#), [getLocation](#), [getRegisteredServices](#), [getResource](#), [getResources](#), [getServicesInUse](#), [getState](#), [getSymbolicName](#), [hasPermission](#), [loadClass](#), [start](#), [start](#), [stop](#), [stop](#), [uninstall](#), [update](#), [update](#)

Field Detail

5.1.1 SECURITY

static final java.lang.String **SECURITY**

The name of a Security Manager class with public empty constructor. A valid value is also true, this means that the framework should instantiate its own security manager. If not set, security could be defined by a parent framework or there is no security. This can be detected by looking if there is a security manager set

See Also:
[Constant Field Values](#)

5.1.2 STORAGE

```
static final java.lang.String STORAGE
```

A valid file path in the file system to a directory that exists. The framework is free to use this directory as it sees fit. This area can not be shared with anything else. If this property is not set, the framework should use a file area from the parent bundle. If it is not embedded, it must use a reasonable platform default.

See Also:

[Constant Field Values](#)

5.1.3 LIBRARIES

```
static final java.lang.String LIBRARIES
```

A list of paths (separated by path separator) that point to additional directories to search for platform specific libraries

See Also:

[Constant Field Values](#)

5.1.4 EXECPERMISSION

```
static final java.lang.String EXECPERMISSION
```

The command to give a file executable permission. This is necessary in some environments for running shared libraries.

See Also:

[Constant Field Values](#)

5.1.5 ROOT_CERTIFICATES

```
static final java.lang.String ROOT_CERTIFICATES
```

Points to a directory with certificates. ###??? Keystore? Certificate format?

See Also:

[Constant Field Values](#)

5.1.6 WINDOWSYSTEM

```
static final java.lang.String WINDOWSYSTEM
```

Set by the configurator but the framework should provide a reasonable default.

See Also:

[Constant Field Values](#)

Method Detail

5.1.7 `init`

```
void init(java.util.Properties configuration)
```

Configure this framework with the given properties. These properties can contain framework specific properties or of the general kind defined in the specification or in this interface.

Parameters:

`configuration` - The properties. This properties can be backed by another properties, it can there not be assumed that it contains all keys. Use it only through the `getProperty` methods. This parameter may be null.

5.1.8 `waitForStop`

```
void waitForStop(long timeout)  
    throws java.lang.InterruptedException
```

Wait until the framework is completely finished. This method will return if the framework is stopped and has cleaned up all the framework resources.

Parameters:

`timeout` - Maximum number of milliseconds to wait until the framework is finished. Specifying a zero will wait indefinitely.

Throws:

`java.lang.InterruptedException` - When the wait was interrupted

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: NESTED | [FIELD](#) | CONSTR | [METHOD](#)

DETAIL: [FIELD](#) | CONSTR | [METHOD](#)

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: NESTED | [FIELD](#) | CONSTR | [METHOD](#)

DETAIL: [FIELD](#) | CONSTR | [METHOD](#)

5.2 org.osgi.service.command Interface **CommandProcessor**

```
public interface CommandProcessor
```

A Command Processor is a service that is registered by a script engine that can execute commands. A Command Processor is a factory for Command Session objects. The Command Session maintains execution state and holds the console and keyboard streams. A Command Processor must track any services that are registered with the `COMMAND_SCOPE` and `COMMAND_FUNCTION` properties. The functions listed in the `COMMAND_FUNCTION` property must be made available as functions in the script language. TODO The javadoc in this class need a good scrub before release.

Version:

\$Revision: 5214 \$

Field Summary

static java.lang.String	COMMAND_FUNCTION A String, array, or list of method names that may be called for this command provider.
static java.lang.String	COMMAND_SCOPE The scope of commands provided by this service.

Method Summary

CommandSession	createSession (java.io.InputStream in, java.io.PrintStream out, java.io.PrintStream err) Create a new command session associated with IO streams.
--------------------------------	--

Field Detail

5.2.1 `COMMAND_SCOPE`

```
static final java.lang.String COMMAND_SCOPE
```

The scope of commands provided by this service. This name can be used to distinguish between different command providers with the same function names.

See Also:

[Constant Field Values](#)

5.2.2 COMMAND_FUNCTION

```
static final java.lang.String COMMAND_FUNCTION
```

A String, array, or list of method names that may be called for this command provider. A name may end with a *, this will then be calculated from all declared public methods in this service. Help information for the command may be supplied with a space as separation.

See Also:

[Constant Field Values](#)

Method Detail

5.2.3 createSession

```
CommandSession createSession(java.io.InputStreamin,  
                                java.io.PrintStreamout,  
                                java.io.PrintStreamerr)
```

Create a new command session associated with IO streams. The session is bound to the life cycle of the bundle getting this service. The session will be automatically closed when this bundle is stopped or the service is returned. The shell will provide any available commands to this session and can set additional variables.

Parameters:

`in` - The value used for System.in
`out` - The stream used for System.out
`err` - The stream used for System.err

Returns:

A new session.

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.3 org.osgi.service.command Interface CommandSession

public interface **CommandSession**

A Command Session holds the executable state of a script engine as well as the keyboard and console streams. A Command Session is not thread safe and should not be used from different threads at the same time. TODO The javadoc in this class need a good scrub before release.

Version:

\$Revision: 5214 \$

Method Summary

void	close () Close this command session.
java.lang.Object	convert (java.lang.Class type, java.lang.Object instance) Convert an object to another type.
java.lang.Object	execute (java.lang.CharSequence commandline) Execute a program in this session.
java.lang.Object	execute (java.lang.CharSequence commandline, java.io.InputStream in, java.io.OutputStream out, java.io.OutputStream err) Execute a program in this session but override the different streams for this call only.
java.lang.CharSequence	format (java.lang.Object target, int level) Convert an object to string form (CharSequence).
java.lang.Object	get (java.lang.String name) Get the value of a variable.
java.io.OutputStream	getConsole () Return the OutputStream for the console.
java.io.InputStream	getKeyboard () Return the input stream that is the first of the pipeline.
void	put (java.lang.String name, java.lang.Object value) Set the value of a variable.

Method Detail

5.3.1 execute

java.lang.Object **execute** (java.lang.CharSequence commandline)
throws java.lang.Exception

Execute a program in this session.

Parameters:

commandline - ###

Returns:

the result of the execution

Throws:

java.lang.Exception - ###

5.3.2 execute

```
java.lang.Object execute(java.lang.CharSequencecommandline,  
                           java.io.InputStreamin,  
                           java.io.PrintStreamout,  
                           java.io.PrintStreamerr)  
throws java.lang.Exception
```

Execute a program in this session but override the different streams for this call only.

Parameters:

commandline -
in - ###
out - ###
err - ###

Returns:

the result of the execution

Throws:

java.lang.Exception - ###

5.3.3 close

```
void close()
```

Close this command session. After the session is closed, it will throw IllegalStateException when it is used.

5.3.4 getKeyboard

```
java.io.InputStream getKeyboard()
```

Return the input stream that is the first of the pipeline. This stream is sometimes necessary to communicate directly to the end user. For example, a "less" or "more" command needs direct input from the keyboard to control the paging.

Returns:

InputStream used closest to the user or null if input is from a file.

5.3.5 getConsole

```
java.io.PrintStream getConsole()
```

Return the PrintStream for the console. This must always be the stream "closest" to the user. This stream can be used to post messages that bypass the piping. If the output is piped to a file, then the object returned must be null.

Returns:
###

5.3.6 get

```
java.lang.Object get(java.lang.Stringname)
```

Get the value of a variable.

Parameters:
name - ###
Returns:
###

5.3.7 put

```
void put(java.lang.Stringname,  
         java.lang.Objectvalue)
```

Set the value of a variable.

Parameters:
name - Name of the variable.
value - Value of the variable

5.3.8 format

```
java.lang.CharSequence format(java.lang.Objecttarget,  
                               intlevel)
```

Convert an object to string form (CharSequence). The level is defined in the Converter interface, it can be one of INSPECT, LINE, PART. This function always returns a non null value. As a last resort, toString is called on the Object.

Parameters:
target -
level -
Returns:

###

5.3.9 convert

```
java.lang.Object convert(java.lang.Class type,  
                           java.lang.Object instance)
```

Convert an object to another type.

Parameters:

type - ###
instance - ###

Returns:

###

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.4 org.osgi.service.command Interface Converter

```
public interface Converter
```

A converter is a service that can help create specific object types from a string, and vice versa. The shell is capable of coercing arguments to their proper type. However, sometimes commands require extra help to do this conversion. This service can implement a converter for a number of types. The command shell will rank these services in order of service.ranking and will then call them until one of the converters succeeds. TODO The javadoc in this class need a good scrub before release.

Version:

\$Revision: 5214 \$

Field Summary

static java.lang.String	CONVERTER_CLASSES This property is a string, or array of strings, and defines the classes or interfaces that this converter recognizes.
static int	INSPECT Print the object in detail.
static int	LINE Print the object as a row in a table.
static int	PART Print the value in a small format so that it is identifiable.

Method Summary

java.lang.Object	convert (java.lang.Class desiredType, java.lang.Object in) Convert an object to the desired type.
java.lang.CharSequence	format (java.lang.Object target, int level, Converter escape) Convert an object to a CharSequence object in the requested format.

Field Detail

5.4.1 CONVERTER_CLASSES

static final java.lang.String **CONVERTER_CLASSES**

This property is a string, or array of strings, and defines the classes or interfaces that this converter recognizes. Recognized classes can be converted from a string to a class and they can be printed in 3 different modes.

See Also:
[Constant Field Values](#)

5.4.2 INSPECT

static final int **INSPECT**

Print the object in detail. This can contain multiple lines.

See Also:
[Constant Field Values](#)

5.4.3 LINE

static final int **LINE**

Print the object as a row in a table. The columns should align for multiple objects printed beneath each other. The print may run over multiple lines but must not end in a CR.

See Also:

[Constant Field Values](#)

5.4.4 PART

```
static final int PART
```

Print the value in a small format so that it is identifiable. This printed format must be recognizable by the conversion method.

See Also:

[Constant Field Values](#)

Method Detail

5.4.5 convert

```
java.lang.Object convert(java.lang.ClassdesiredType,  
                           java.lang.Objectin)  
    throws java.lang.Exception
```

Convert an object to the desired type. Return null if the conversion can not be done. Otherwise return and object that extends the desired type or implements it.

Parameters:

`desiredType` - The type that the returned object can be assigned to

`in` - The object that must be converted

Returns:

An object that can be assigned to the desired type or null.

Throws:

`java.lang.Exception`

5.4.6 format

```
java.lang.CharSequence format(java.lang.Objecttarget,  
                                intlevel,  
                                Converterescape)  
    throws java.lang.Exception
```

Convert an object to a CharSequence object in the requested format. The format can be INSPECT, LINE, or PART. Other values must throw IllegalArgumentException.

Parameters:

`target` - The object to be converted to a String

`level` - One of INSPECT, LINE, or PART.

escape - Use this object to format sub ordinate objects.

Returns:

A printed object of potentially multiple lines

Throws:

java.lang.Exception

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.5 org.osgi.service.command Interface Function

```
public interface Function
```

A Function is a a block of code that can be executed with a set of arguments, it returns the result object of executing the script. TODO The javadoc in this class need a good scrub before release.

Version:

\$Revision: 5214 \$

Method Summary

java.lang.Object	execute (CommandSession session, java.util.List arguments) Execute this function and return the result.
------------------	---

Method Detail

5.5.1 execute

```
java.lang.Object execute (CommandSession session,  
                           java.util.List arguments)  
    throws java.lang.Exception
```

Execute this function and return the result.

Parameters:

session - ###
arguments - ###

Returns:

the result from the execution.

Throws:

java.lang.Exception - if anything goes terribly wrong

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.6 org.osgi.service.threadio Interface ThreadIO

```
public interface ThreadIO
```

Enable multiplexing of the standard IO streams for input, output, and error. This service guards the central resource of IO streams. The standard streams are singletons. This service replaces the singletons with special versions that can find a unique stream for each thread. If no stream is associated with a thread, it will use the standard input/output that was originally set. TODO The javadoc in this class need a good scrub before release.

Version:

\$Revision: 5214 \$

Method Summary	
void	close () Cancel the streams associated with the current thread.
void	setStreams (java.io.InputStream in, java.io.OutputStream out, java.io.PrintStream err) Associate this streams with the current thread.

Method Detail

5.6.1 setStreams

```
void setStreams(java.io.InputStream in,  
               java.io.PrintStream out,  
               java.io.PrintStream err)
```

Associate this streams with the current thread. Ensure that when output is performed on System.in, System.out, System.err it will happen on the given streams. The streams will automatically be canceled when the bundle that has gotten this service is stopped or returns this service.

Parameters:

`in` - InputStream to use for the current thread when System.in is used
`out` - PrintStream to use for the current thread when System.out is used
`err` - PrintStream to use for the current thread when System.err is used

5.6.2 close

```
void close()
```

Cancel the streams associated with the current thread. This method will not do anything when no streams are associated.

[Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

6 Alternatives

Maybe we should move the syntax to this section. Because we have standard launching the necessity of a standard syntax has become less. Hmm.

6.1 Considered setParentBundle

This section was denied because it was deemed to premature. An other RFC will look at nested frameworks.

setParentBundle(Bundle) – If a framework is embedded in another framework, then it must give the child framework the bundle object of its representation in the parent. That is, if you embed a framework in an OSGi framework, the parent is the bundle object of the code that manages the embedding. This bundle must be registered in the service registry as a Bundle service with the property: org.osgi.framework.parent=true. Singleton services like thread IO and URL handlers should use this service to synchronize their behavior with the ancestor frameworks. This method can be repeatedly called when the framework is not started.

7 Security Considerations

Obviously, a shell language provides ample opportunities for malice. In principle, anything in the system is accessible, just like from Java. The protection against malicious behavior is based up the Java 2 security model. This allows the shell and all commands to be ignorant of any security issues, unless they want to perform operations that they have access to but a potential user has not. Such code must be executed in a doPrivileged block.

The IO processors have the responsibility for protecting against malicious users.

should copy some of the text of DMT Admin because it follows the same procedures

8 Document Support

8.1 References

- [1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
- [2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

8.2 Author's Address

Name	Peter Kriens
Company	aQute
Address	9c, Avenue St. Drezerie
Voice	+33 633982260
e-mail	Peter.Kriens@aQute.biz

8.3 Acronyms and Abbreviations

8.4 End of Document

RFC 0134 Declarative Services Update

Draft

8 Pages

Abstract

This RFC specifies some minor changes requested for Declarative Services.

Copyright © IBM Corporation 2008

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.....	3
1 Introduction	3
2 Application Domain	3
3 Technical Solution	4
3.1 Bug 144: No component instance if no Configuration.....	4
3.2 Bug 244: Extend SCR to allow alternate activate and deactivate method signatures.....	4
3.2.1 Component deactivation reasons.....	5
3.3 Bug 567: Allow use of wildcards in Service-Component header	6
3.4 Bug 600: Making name attributes optional.....	6
3.5 XML schema namespace change.....	7
4 Considered Alternatives	7
5 Security Considerations	7
6 Document Support	8
6.1 References.....	8
6.2 Author's Address.....	8
6.3 Acronyms and Abbreviations.....	8
6.4 End of Document.....	8

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 6.1.

Source code is shown in this typeface.

0.3 Revision History

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

Revision	Date	Comments
Initial	2008 Mar 11	Initial Draft BJ Hargrave, hargrave@us.ibm.com
2 nd draft	2008 Mar 16	Updated based upon feedback from CPEG BJ Hargrave, hargrave@us.ibm.com
3 rd draft	2008 Mar 27	Updated based upon feedback from CPEG BJ Hargrave, hargrave@us.ibm.com
4 th draft	2008 Apr 24	Updated based upon bug 600. BJ Hargrave, hargrave@us.ibm.com

1 Introduction

Some minor updates to the Declarative Services Specification have been requested since it was released. This RFC specifies those changes.

2 Application Domain

This RFC defines changes to section 112, Declarative Services, of the specification.

3 Technical Solution

3.1 Bug 144: No component instance if no Configuration

A way is needed for the component declaration to say only create a component configuration IF there is a Configuration(or Configurations).

To support this, the follow attribute is added to the component element:

```
<attribute name="configuration-policy" type="scr:Tconfiguration-policy"
          default="optional" use="optional" />
<simpleType name="Tconfiguration-policy">
  <restriction base="string">
    <enumeration value="optional" />
    <enumeration value="require" />
    <enumeration value="ignore" />
  </restriction>
</simpleType>
```

If the attribute is present and set to require, then a component cannot be satisfied (section 112.5.2) unless there is a Configuration in ConfigurationAdmin for the component. In this situation, the *No Configuration* case in 112.7 does not apply. If the component is a Factory Component and the component is not satisfied because there is no Configuration present, then the ComponentFactory service will not be registered.

If the attribute is present and set to ignore, then ConfigurationAdmin will not be consulted for the component. In this situation, only the *No Configuration* case in 112.7 applies.

If the attribute is not present or present and set to optional, then SCR will act as it did prior to this RFC. That is, a Configuration will be used if present in ConfigurationAdmin.

3.2 Bug 244: Extend SCR to allow alternate activate and deactivate method signatures

A way is needed to avoid using DS API at all in components with SCR. This means the activate and deactivate methods should not require the ComponentContext parameter. We should also allow the names of the activate and deactivate methods to be specified to avoid requiring specific method names.

To support this, the follow attributes will be added to the component element:

```
<attribute name="activate" type="token" use="optional" default="activate" />
<attribute name="deactivate" type="token" use="optional" default="deactivate" />
```

The activate attribute will specify the name of the activate method and the deactivate attribute will specify the name of the deactivate method.

The signature for the activate and deactivate methods is:

```
protected void <method-name>(<arguments>);
```

<arguments> can be zero or more arguments.

For the activate method each argument must be of one of the following types:

- `ComponentContext` – the Component Context for the component
- `BundleContext` – the Bundle Context of the component's bundle

If any argument of the activate method is not one of the above types, SCR must log an error message with the Log Service, if present, and the component configuration is not activated.

For the deactivate method each argument must be of one of the following types:

- `int/Integer` – the deactivation reason
- `ComponentContext` – the Component Context for the component
- `BundleContext` – the Bundle Context of the component's bundle

If any argument of the deactivate method is not one of the above types, SCR must log an error message with the Log Service, if present, and the deactivation of the component configuration will continue.

The methods may also be declared public. The same rules as specified in 112.5.8 will be used to locate the activate and deactivate methods in the implementation class hierarchy.

3.2.1 Component deactivation reasons

When a component is deactivated, the reason for the deactivation can be passed to the deactivate method. The following deactivation reasons are specified in `ComponentConstants`.

```
/**
 * The reason the component instance was deactivated is unspecified.
 *
 * @since 1.1
 */
public static final int DEACTIVATION_REASON_UNSPECIFIED = 0;

/**
 * The component instance was deactivated because the component was
disabled.
 *
 * @since 1.1
 */
public static final int DEACTIVATION_REASON_DISABLED = 1;

/**
 * The component instance was deactivated because a reference became
unsatisfied.
 *
 * @since 1.1
 */
public static final int DEACTIVATION_REASON_REFERENCE = 2;
```

```

/**
 * The component instance was deactivated because its configuration was
changed.
 *
 * @since 1.1
 */
public static final int DEACTIVATION_REASON_CONFIGURATION_MODIFIED = 3;

/**
 * The component instance was deactivated because its configuration was
deleted.
 *
 * @since 1.1
 */
public static final int DEACTIVATION_REASON_CONFIGURATION_DELETED = 4;

/**
 * The component instance was deactivated because the component was
disposed.
 *
 * @since 1.1
 */
public static final int DEACTIVATION_REASON_DISPOSED = 5;

/**
 * The component instance was deactivated because the bundle was stopped.
 *
 * @since 1.1
 */
public static final int DEACTIVATION_REASON_BUNDLE_STOPPED = 6;

```

3.3 Bug 567: Allow use of wildcards in Service-Component header

A way is needed to allow wild specification of the XML documents containing the component descriptions.

To support this, section 112.4.1 will be updated to state that the path element of the Service-Component header grammar may include wildcards in the last component of the path. For example:

```
Service-Component: OSGI-INF/*.xml
```

Only the last component of the path may use wildcards so that `Bundle.findEntries` can be used to locate the XML document within the bundle and its fragments.

3.4 Bug 600: Making name attributes optional

To reduce the amount of XML that must be written for a component description, the `name` attributes of the `component` and `reference` elements will be changed from required to optional. This change is only effective for documents in the new namespace.

Draft

16 May 2008

The default value of the `name` attribute of the `component` element is the value of the `class` attribute of the nested `implementation` element.

The default value of the `name` attribute of the `reference` element is the value of the `interface` attribute of the `reference` element.

3.5 XML schema namespace change

The schema namespace is updated to

<http://www.osgi.org/xmlns/scr/v1.1.0>

This is due to changes to support backwards and forwards computability in the future (after this namespace change). Hopefully we can avoid further namespace changes by introducing extra requirements on the SCR parser at this time.

4 Considered Alternatives

None at this time.

5 Security Considerations

These changes do not affect the security of the specification.

6 Document Support

6.1 References

- [1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
- [2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

6.2 Author's Address

Name	BJ Hargrave
Company	IBM Corporation
Address	800 N Magnolia Av, Orlando, FL 32803
Voice	+1 386 848 1781
e-mail	hargrave@us.ibm.com

6.3 Acronyms and Abbreviations

6.4 End of Document



Enterprise Design Documents

OSGi Service Platform Release 4

Version 4.2 - Early Draft

Revision 1.0
5 August 2008



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.

Copyright © OSGi Alliance 2008.

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	3
0.3 Revision History	3
1 Introduction	4
2 Application Domain	4
2.1 Crash Recovery	5
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	10
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	11
4.5.5 Commit Resource	11
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	12

4.6.4 Performance12

4.6.5 Management of Transaction13

4.6.6 Heuristic Exceptions13

4.6.7 Examples13

5 Security Considerations14

5.1 Imposing as Transaction Manager14

5.2 Transaction Permission14

6 Document Support15

6.1 References.....15

6.2 Author’s Address15

6.3 Acronyms and Abbreviations16

6.4 End of Document16

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.

Revision	Date	Comments
Initial	Aug 30 2004	Peter Kriens
rewriting	Oct 30 2007	Pavlin Dobrev
0.2	Oct 31 2007	Apply comments from Valentin Valchev
0.3	Nov 08 2007	Pavlin Dobrev – minor misspellings
0.4	Nov 22 2007	Peter Kriens – edit and added comments
0.5	Nov 29 2007	Pavlin Dobrev – answer on some comments
0.6	Dec 13 2007	Pavlin Dobrev – answer to Thomas Watson <tjwatson@us.ibm.com> comments
0.7	Jul 09 2008	Ian Robinson/Roman Roelofsen – rebase on JTA.
0.71	6 Aug 2008	Prepare for public draft.

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* – A transaction's changes to the state are atomic: either all happen or none happen. 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* – Even though transactions execute concurrently, it appears to each transaction, T, that others executed either before or after T, but never both.
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 centralized 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 any resource locks 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 CORBA Object Transaction Service (OTS) [4] and WS-AtomicTransaction [5] defines mechanisms by which transaction contexts are implicitly propagated on remote requests over IIOB and SOAP/HTTP transports to be used by the transaction manager in the target system to add the work of the thread in the target system to the overall distributed global 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 under 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.. In the 2PC model the transaction manager asks each resource manager its opinion during the initial *prepare* phase. If any resource manager indicates that there was an error, the transaction is *rolled back*. Rolling back means that all changes done inside the transaction are undone. If all the resource managers respond positively to the prepare request then the transaction manager directs all the resource managers to commit their updates in the second phase of 2PC. The resource managers must then make their changes permanent. Resource managers that provide transaction-based locking and isolation then release locks and make updates visible outside the transaction.

The *2-phase commit* model accommodates the possibility that a resource manager may not be able to honour its responsibility to commit after it has successfully prepared via a *heuristics* protocol. This indicates a failure of global atomicity and typically requires administrative intervention to restore data integrity.

Transactions are not limited to changes in a database. Transactions apply equally well to sensors and actuators, messages and other actions.

2.1 Crash Recovery

An important aspect of transaction atomicity is *crash recovery*. A crash is when a program or subsystem (for example transaction manager or resource manager) unexpectedly dies 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 a transaction is prepared.

Recovery processing is required following a crash to resolve any parts of a global transaction that were prepared but not completed at the point when the crash occurred. Resource managers with prepared work may be holding

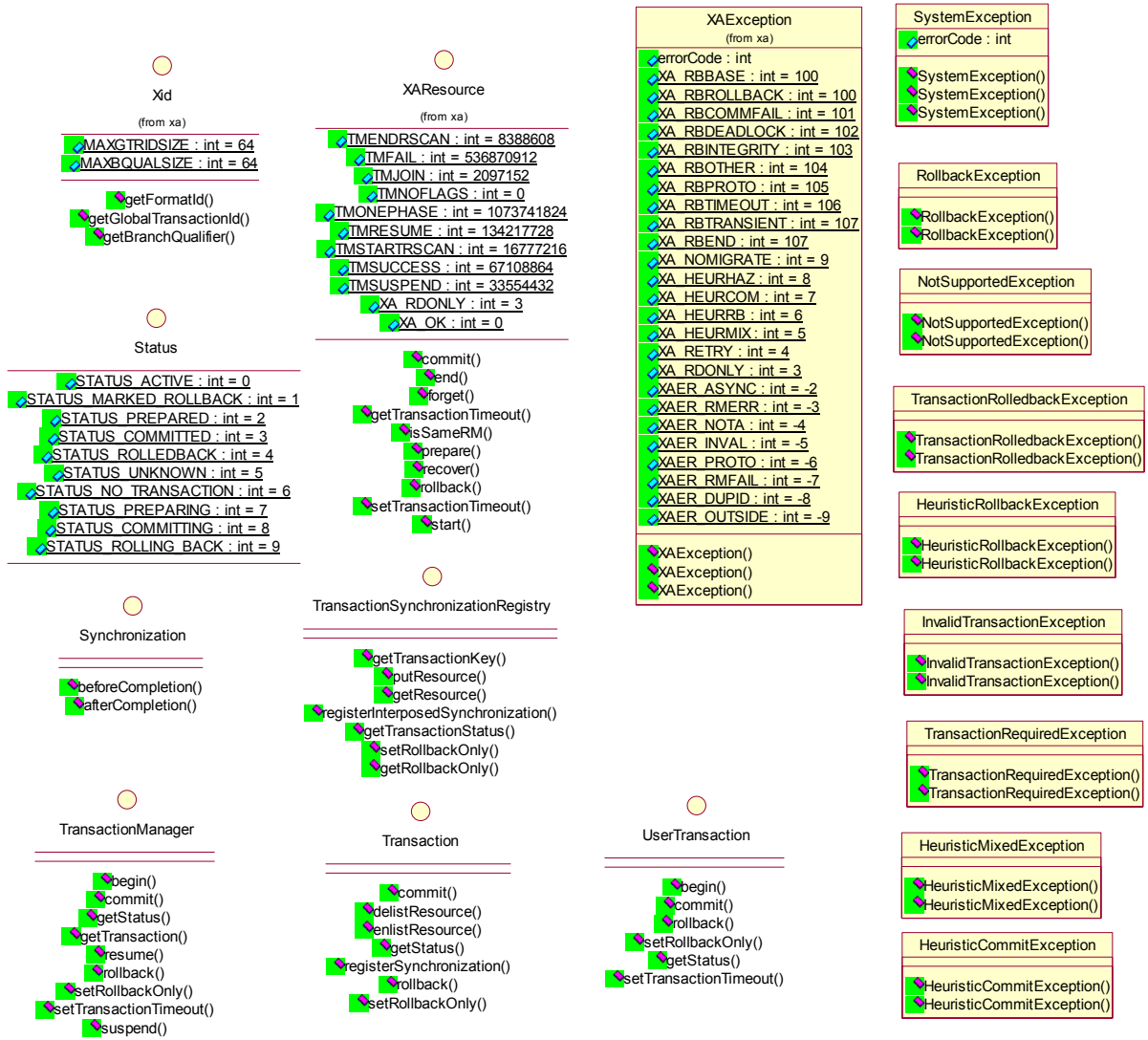
locks and need to be directed to commit or rollback their work. The XA 2PC protocol defines a recovery protocol between the TM and RMs to resolve such “in-doubt” work.

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 `javax.transaction.xa` package which contains the `XAResource` interface. The model in JTA is that resource managers (`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.



Class Diagram of javax.transaction API

2.3 Why Transactions

The key reason to support transactions is: simplicity. Decades of experience in the IT world have proven that transactions concentrate complexity at the place where it should be: the resource manager. This implies that they significantly simplify the work of the application programmer. Transactions provide a simple and clean model of the world with excellent exception handling. Transactions provide a framework so that programmers only have to think in consistent states and do not have to handle the messy issues related to partially completed operations and concurrent access.

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 and 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 crash.

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 the JTA XAResource interface is already implemented by many providers of Java resource adapters. 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 additionally 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 service and **MUST** implement the `javax.transaction.UserTransaction`, `javax.transaction.TransactionManager`, `javax.transaction.Transaction` and `javax.transaction.SynchronizationRegistry` interfaces as defined in [6].
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 `SynchronizationRegistry.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.enlist(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.
- Nested transactions are not supported. You are not allowed to run a transaction within another one.

4.3.2 Transaction Manager

The transaction manager provides transactional capabilities for the framework. It

- Creates transactions and associates them with the current thread of the originator application.
- Accepts enlistment of volatile and transactional resources.
- 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 crash to ensure the atomic completion of transactional work.
- An OSGi transaction service implementation represents a specific transaction with an object that implements the `javax.transaction.Transaction` interface and it is this object with which transactional resources are enlisted. A `Transaction` object is obtained for the current transaction from the `getTransaction()` method of the `TransactionManager` interface.

4.3.3 Volatile Resources

Volatile resources are components which 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. 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 once resource work is performed under 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` and `TransactionSynchronizationRegistry` interfaces in the OSGi service registry, using the names “`javax.transaction.UserTransaction`” and “`javax.transaction.TransactionSynchronizationRegistry`” respectively. An OSGi transaction service implementation MUST also register a service object for the `TransactionManager` interface in the service registry using the name “`javax.transaction.TransactionManager`” but MAY put restrictions on which bundles can use this service object.

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

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 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 `enlist(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 commit or rollback processing via the `UserTransaction` interface. During commit processing, if at least one of the resources participating in the transaction fails to perform the required operations, the transaction is rolled back.

4.5.4 Prepare Resource

The transactional manager uses the “two phase commit” pattern to ensure the consistent state of the system. When the originator requests a commit of the transaction, the TM calls on each participating resource their “prepare” method. In this stage, the resource provisionally performs any updates and decides whether it make a commitment to 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 `Transaction Resource`. The resource manager is responsible for providing any changes to data visibility that result from the transactional isolation level of the updates and for making the updates the changes persistent.

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, restores the original state of the system as it was before starting the transaction.

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 on each resource their “rollback” method. This method discards any provisional updates within the transaction and so restores the original state of the system.

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 originor. In between, the transaction is managed by a transaction manager. 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 Atom 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 Consistency of the data changes is provided by the application and the resource managers.

4.6.3 End of Transaction

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

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

```
UserManager tx = null;
// UserTransaction can be received from Service Registry

// begin transaction
tx.begin();

// perform some operations in the context of the transaction
try {
    Configuration x = config.createConfiguration("abc");
    x.put("prop", "value");
    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

```
class ConfigResource implements XAResource {
    Transaction t;

    public Configuration createConfiguration(String pid) {
        if( t != null ) {
            t.enlist(this)
            // Transactional operation
            addLog("createConfiguration", pid);
        } else {
            // TODO: non-transactional operation
        }
    }

    public Configuration[] listConfigurations() {
        Configuration ret[] = null;

        if( t != null ) {
            // TODO: add the configurations that are still not committed in CURRENT TRANSACTION
            // This is because all changes needs to be visible in the current transaction
        }
    }
}
```

```
    }

}

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

public void commit(Xid xid) {
    // 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 Imposing as Transaction Manager

The transaction manager has a very central role and it is paramount that no bundle except the intended one can register a Transaction Manager service. This is achieved with ServicePermission REGISTER for this service.

5.2 Transaction Permission

The TransactionPermission defines the security roles required to retrieve or start a transaction.

-

6 Document Support

6.1 References

- [1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
- [2]. Transaction Processing, Jim Gray and Andreas Reuter. Morgan Kaufmann Publishers, ISBN 1.55860-190-2
- [3]. Distributed Transaction Processing: The XA+ Specification Version 2, The Open Group, ISBN: 1-85912-046-6
- [4]. Object Transaction Service v1.4, OMG, <http://www.omg.org/cgi-bin/doc?formal/2003-09-02>
- [5]. WS-AtomicTransaction v1.1, OASIS, <http://docs.oasis-open.org/ws-tx/wsat/2006/06>
- [6]. JTA Specification v1.1, <http://java.sun.com/products/jta/>

6.2 Author's Address

Name	Peter Kriens
Company	aQute
Address	9C, Avenue St. Drézéry
Voice	+15123514821
e-mail	Peter.kriens@aQute.biz

Name	Pavlin Dobrev
Company	ProSyst Software GmbH
Address	Dürener Str. 405, 50858 Cologne, Germany
Voice	+49 221 6604-0
e-mail	p.dobrev@prosyst.com

6.3 Acronyms and Abbreviations

6.4 End of Document



RFC 119 - Distributed OSGi

47 Pages

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.

Copyright © OSGi Alliance 2008.

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	4
0.3 Revision History	4
1 Introduction	5
1.1 Open Items	6
1.2 Terminology	6
1.3 List of Symbols.....	7
2 Application Domain	8
3 Problem Description.....	9
3.1 From RFPs 79 & 88:	9
3.2 Scenario diagrams	10
3.2.1 Consumer Side	11
3.2.2 Provider Side	12
3.2.3 A non-OSGi distributed client using an OSGi service	13
3.2.4 An OSGi client using a remote non-OSGi Service	13
3.3 Roles.....	14
3.3.1.1 Solution Architect.....	16
3.3.1.2 Component Designer.....	16
3.3.1.3 Developer	16
3.3.1.4 Assembler.....	16
3.3.1.5 Solution Deployer	17
3.3.1.6 Testing.....	17
3.3.1.7 Runtime (Framework).....	17
4 Requirements.....	17
4.1 From RFP 79	17
4.2 From RFP 88	18
4.3 Further requirements	20
4.3.1 Levels of transparency.....	20
5 Technical Solution.....	20
5.1 Overview of contributions to the OSGi standard	20
5.1.1 Summary of Changes to the OSGi Core	20
5.1.2 Summary of Additional Services.....	21

5.2 Distribution software	21
5.2.1 Functionality	21
5.2.2 Interface description.....	23
5.2.2.1 Distribution Software Interface	23
5.2.2.2 Exception Handling.....	24
5.3 Discovery Service	25
5.3.1 Functionality	25
5.3.2 Discovery using a local file(s)	26
5.3.3 Discovery Service Federation and Interworking	27
5.3.4 Useful Discovery Service Properties	28
5.3.5 Interface description.....	28
5.3.5.1 Java interface description.....	28
5.4 Service Registry Hooks	32
5.4.1 Registration of Remote Services in Local Service Registry	32
5.4.2 Additional filtering.....	32
5.5 Service Programming Model	32
5.5.1 Service interface description.....	33
5.5.2 Properties.....	33
5.5.2.1 Definition of new Properties.....	33
5.5.2.2 Standard Properties.....	33
5.5.3 Intents	34
5.5.3.1 Example of using Intents	35
5.5.3.2 Defining Intents.....	35
5.5.3.3 OSGi-defined Intents	35
5.5.3.4 SCA-defined Intents	36
5.5.3.5 Qualified Intents.....	37
5.5.3.6 Publishing of Qualified Intents	37
5.5.4 Configuration type.....	37
5.6 Collaboration of new and changed entities.....	38
5.6.1 Interactions on the service provider side	38
5.6.1.1 Exposing a Service remotely	38
5.6.1.2 Service Unregistration	39
5.6.2 Interactions on the service consumer side	40
5.6.2.1 Lookup for a remote Service	40
5.6.2.2 Service invocation	40
5.6.3 Interactions with Non-OSGi service providers and consumers	41
5.6.4 Lifecycle dynamics.....	41
5.7 Best Practices	41
5.7.1.1 Runtime (Framework).....	41
5.7.2 Distribution-related limitations on service interface definitions	42
5.7.3 Connector.....	42
5.7.4 Caching.....	42
5.7.5 Automated Service discovery	42
5.7.6 Bundle organization	42
5.7.7 Proxies	43
5.8 Reference Implementation.....	43
5.8.1 Installing Distribution Software in an OSGi platform.....	43
5.9 Reference Implementation based on SCA	43
6 Considered Alternatives	43
6.1.1 Alternative: using simple properties to define service remoting	43
7 Security Considerations	45

8 Document Support46

 8.1 References.....46

 8.2 Author’s Address46

 8.3 Acronyms and Abbreviations47

 8.4 End of Document47

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.

Revision	Date	Comments
Initial	Jul 17, 2007	Initial draft with information based on RFP 88 Eric Newcomer, IONA, eric.newcomer@iona.com David Bosschaert, IONA, davidb@iona.com
0.1	July 27, 2007	Added parts related to RFP 79 (service discovery) Tim Diekmann, Siemens Communications, tim.diekmann@siemens.com
0.2	August 1-8	Added introductory information, incorporated edits, and filled in sections from relevant RFPs.
0.3	September, 2007	Changes following feedback from the August Face-to-Face <i>Tim Diekmann, Siemens Communications,</i> tim.diekmann@siemens.com Eric Newcomer, IONA, eric.newcomer@iona.com David Bosschaert, IONA, davidb@iona.com
0.4	October, 2007	Tim Diekmann, Siemens Communications, Philipp Konradi, Siemens Corporate Technologies
0.5	November, 2007	Tim Diekmann, Siemens Communications, Philipp Konradi, Siemens Corporate Technologies Klaus Kunte, Siemens Enterprise Networks GmbH & Co KG

Revision	Date	Comments
0.6	February, 2008	Changes as a result of discussions at January F2F Deleted comments that seemed resolved or discussed. Eric Newcomer
0.7	March, 2008	Tim Diekmann, Siemens Communications. Accepted all changes.
0.8	May, 2008	Eric Newcomer, accepted changes, incorporated text about intents, cleaned up comments following their resolution. Graham Charters & Philipp Konradi, qualified intents section.
0.9	June, July 2008	Eric Newcomer, further editorial cleanup
0.9.1	July 2008	Tim Diekmann, minor editorial changes, bug 719
0.9.2	July 2008	Eric Newcomer, changes from July F2F & bug list
0.9.3	August 2008	David Bosschaert, changes relating to bugs 689, 729 and 735.

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 straight forwardly 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 Open Items

- See bug list

1.2 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.

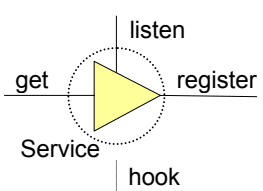
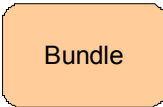
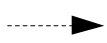



Discovery service: A software entity providing functionality to an *OSGi service platform* that supports the publishing and lookup of services in other address spaces or across machine boundaries, using various existing discovery systems.

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

Service provider: A bundle which provides a service to other service platforms.

1.3 List of Symbols

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

Symbol	Term	Description
	OSGi Service	<ul style="list-style-type: none"> - 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
	OSGi Bundle	<ul style="list-style-type: none"> - provides modularization and encapsulation of components - is a deployment unit (software provisioning) for the OSGi runtime
	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	<ul style="list-style-type: none"> - provides an interface outside of a local OSGi Service Platform - exposing transport or communication protocols, e.g. SOAP/HTTP, CORBA/IOP, RMI, etc.
	Non OSGi Platform	<ul style="list-style-type: none"> - provides a component based environment for enterprise applications - offering non OSGi technologies, e.g. SCA, Spring, etc.
	OSGi Service Platform	<ul style="list-style-type: none"> - 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 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.

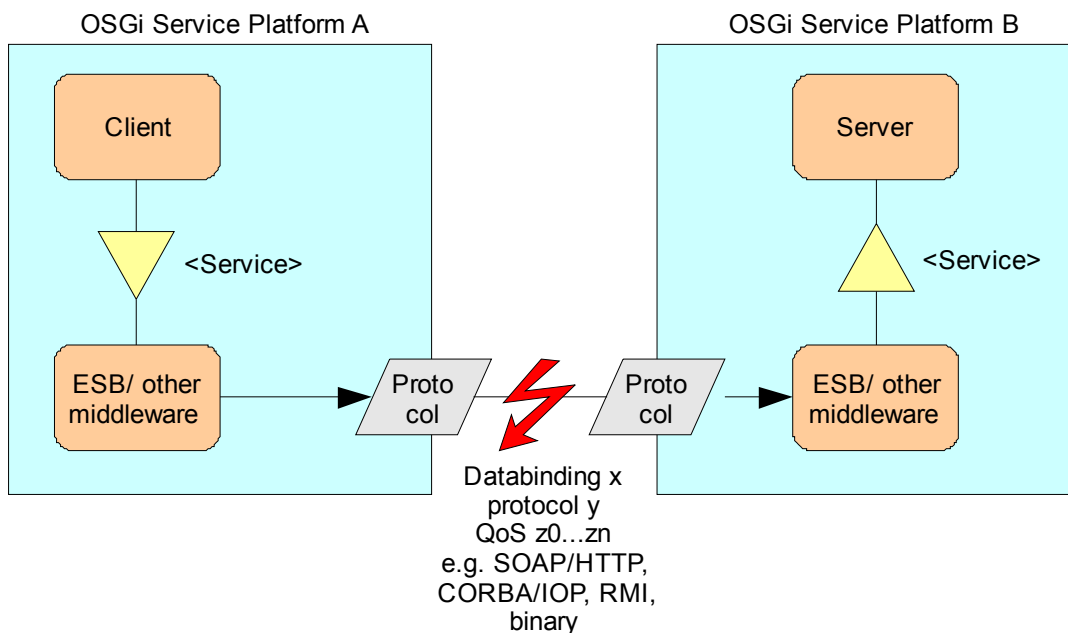


Figure 1 OSGi service consumer using a remote OSGi-service provider

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.

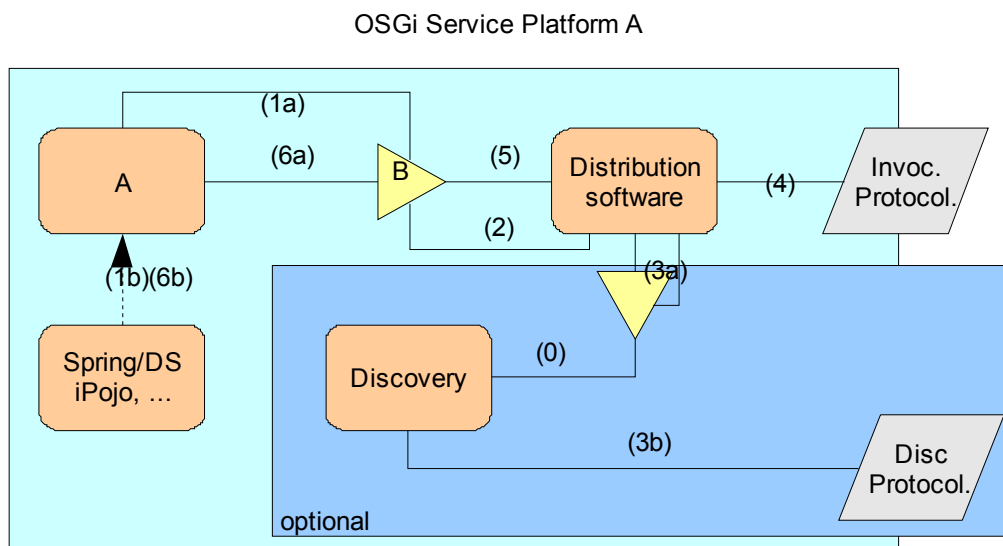


Figure 2 Service Consumer - in OSGi framework

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, Spring 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 service interface of the discovery service to perform the lookup of Service B over the network. The Discovery service is an optional service on the requester side 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.

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.)

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 (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.

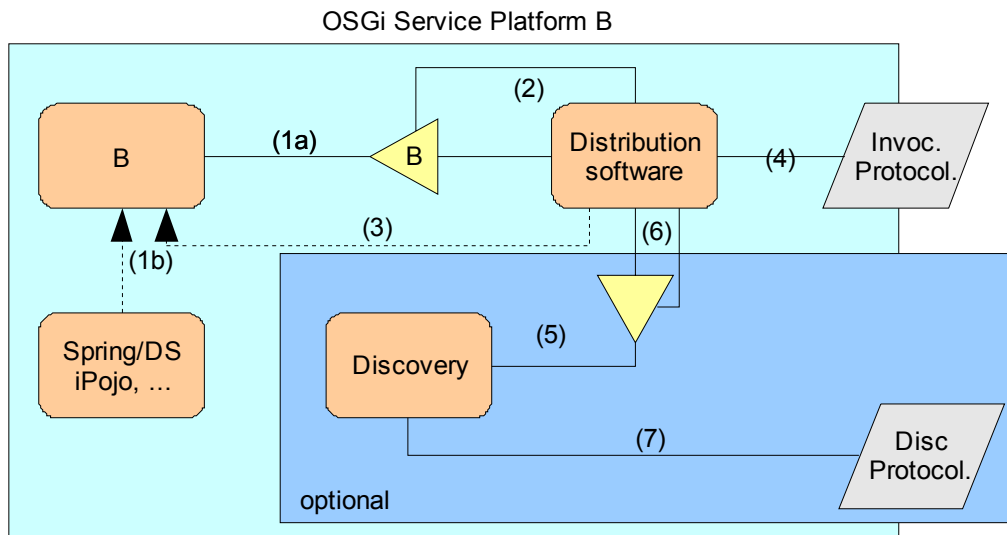


Figure 3: Service provider - OSGi framework

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, Spring, 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 is 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 5 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.

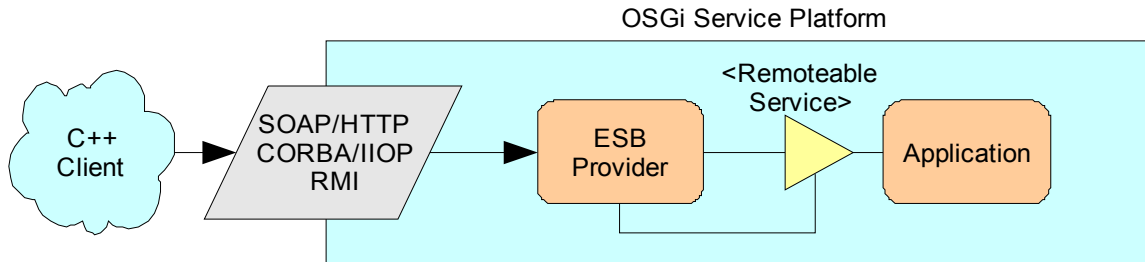


Figure 4 Remote non-OSGi service consumer using an OSGi service provider

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.

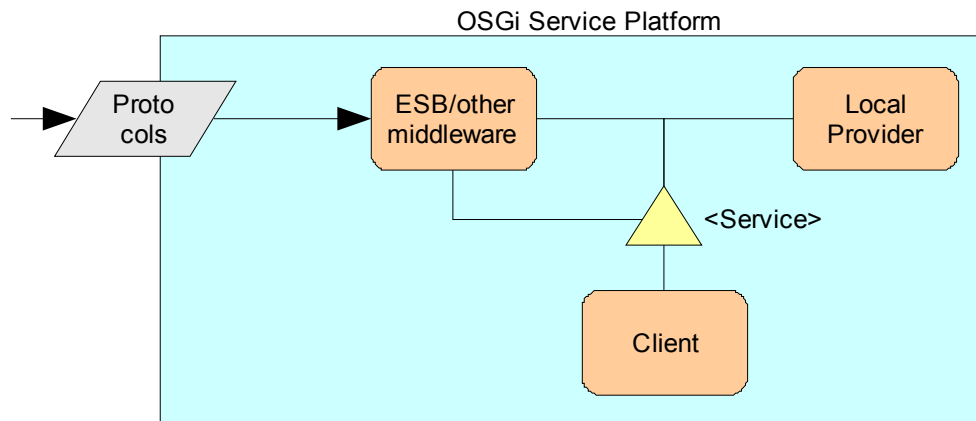


Figure 5 Service consumer uses a distributed but locally available service provider

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).

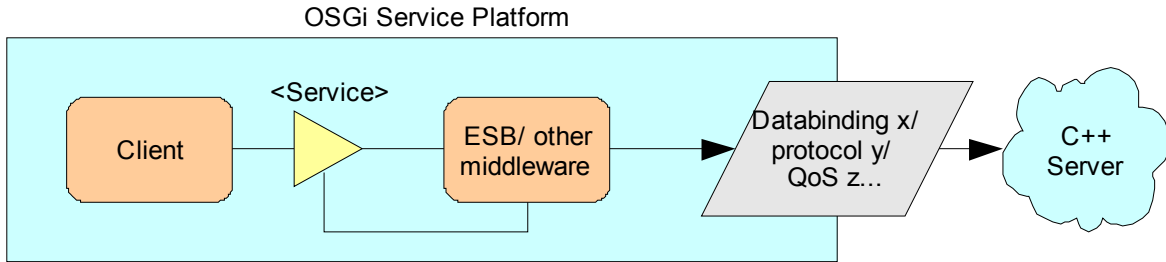


Figure 6 OSGi service consumer using a remote non-OSGi-service provider

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. This section describes the roles relevant to this document.

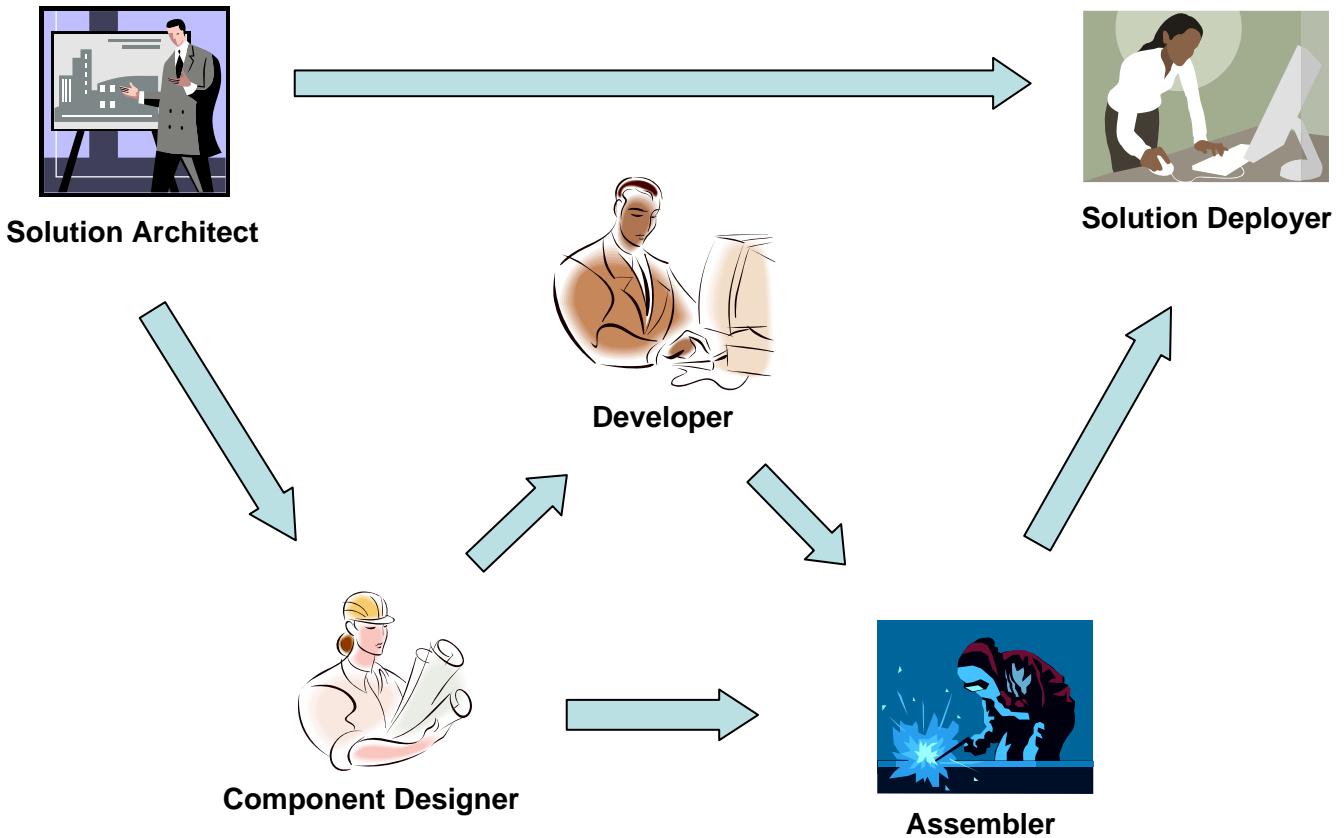


Figure 7, Relationships Among Designer, Developer, Architect, Assembler and Deployer Roles

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:

Role	Required Artifacts	Performed Tasks	Collaborates With	Produced Artifacts
Solution Architect	Application Requirements	Analyses requirements	Solution Deployer	Solution Specification
		Defines SOA architecture	Component Designer	Component Requirements
		Defines the Service Interface and some properties		Service Interface Definition (e.g. UML class diagram, etc.) and property definition (e.g. remoteable)
Component Designer	Component Requirements	Designs the service implementation	Solution Architect	Component Specification
	Service Interface Definition and property definition	Specifies the service interface	Developer Assembler	Service Interface Specification (e.g. WSDL file, IDL file, etc.)
		Defines service-specific properties, intents and optional metadata		Definition of Service Properties, intents and optional metadata (e.g. call by reference, call by value, remoteable, etc.)
Developer	Component Specification	Implements the business logic	Component Designer	Components
	Service Interface Specification	Defines implementation-specific properties	Assembler	
	Definition of Service Properties, intents and optional metadata			
Assembler	Components	Builds installable packages	Components Designer	Bundles
	Definition of Service Properties, intents and optional metadata	Creates service properties (types and <u>defaults</u>)	Developer Solution Deployer	Service Properties, intents and optional metadata (i.e. bundle specific property file or XML file containing defaults)
Solution Deployer	Solution Specification	Deploys and configures the application	Solution Architect	Application
	Bundles	Provides solution specific <u>configuration</u> properties	Assembler	Configuration Properties, intents and optional metadata (e.g. communication protocols, policies, etc.)
	Service Properties, intents and optional metadata			

Note: within a role the members collaborate with each other, e.g. the component designer collaborates with other component designers

3.3.1.1 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.

3.3.1.2 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 'org.osgi.remote.publish', 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.

3.3.1.3 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.

3.3.1.4 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)

3.3.1.5 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.

3.3.1.6 Testing

Testing is part of each role and is accompanied by each produced artifact to ensure performance, robustness and interoperability for the whole solution.

3.3.1.7 Runtime (Framework)

Controls the lifecycle of services and service dependencies (e.g. DS, Spring). 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 IP V6
 - 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 two 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
- Changes to the service programming model for distribution:
 - Reserved properties starting with *org.osgi.* including:
 - *org.osgi.remote.publish* – indicates that the provided service is to be made available remotely, which implies that it is suitable for remote invocations.

- `org.osgi.remote.intents` – list of intents (format and syntax defined by SCA) provided by the component designer and changeable by the deployer.
- `org.osgi.remote.configuration.type` – identifies the metadata type of additional metadata, if any, associated with the service provider or consumer, e.g. “SCA”
- 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

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 to locate OSGi based and non-OSGi services over the network using any available protocol defined by the implementation. A special case of the discovery service may be provided by the local OSGi framework. 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 register it with the discovery service.

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 responsible for interacting with the Discovery Service to register and subsequently discover services that it has made available over the network. On the client side the Distribution Service will also interact with the Discovery Service when it needs extra metadata for creating a proxy to the remote service.

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 distributed software.

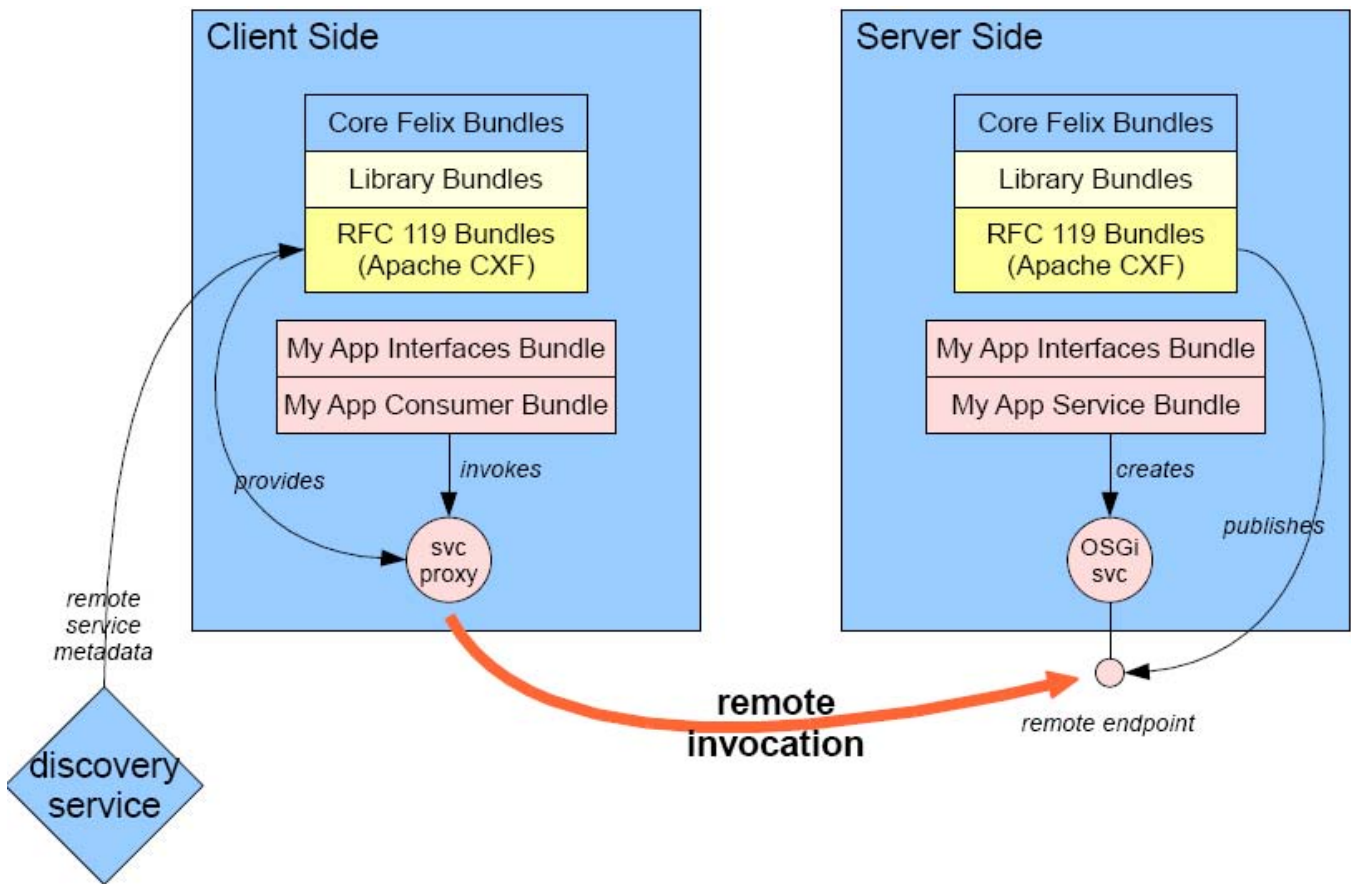


Figure 8, Example Implementation of Distributed OSGi using Web services

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 a remote 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 =
        "org.osgi.remote.distribution.product";

    /**
     * Service Registration property for the version of the Distribution
     * Provider product.
     */
    static final String PROP_KEY_PRODUCT_VERSION =
        "org.osgi.remote.distribution.product.version";

    /**
     * Service Registration property for the Distribution Provider product
     * vendor name.
     */
    static final String PROP_KEY_VENDOR_NAME =
        "org.osgi.remote.distribution.vendor";

    /**
     * Service Registration property that lists the intents supported by this
     * DistributionProvider.
     */
    static final String PROP_KEY_SUPPORTED_INTENTS =
        "org.osgi.remote.distribution.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 array is returned.
 */
ServiceReference[] getRemoteServices();

/**
 * @return ServiceReferences of local services that are exposed remotely
 * using this DisitributionProvider. 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
 * array is returned.
 */
ServiceReference[] getExposedServices();

/**
 * @return Local ServiceReferences of exposed services that are published
 * remotely to a discovery mechanism using this DisitributionProvider.
 * Note that certain services might be exposed without being published.
 * This API returns all the published service. If no services are
 * published an empty array is returned.
 */
ServiceReference[] getPublishedServices();

/**
 * Provides access to extra properties set by the DistributionProvider on
 * client side proxies given an exposed ServiceReference. These properties
 * are not available on the server-side ServiceReference of the published
 * service but will be on the remote client side proxy to this service.
 * This API provides access to these extra properties from the publishing side.
 * E.g. a service is exposed over SOAP and HTTP. Because of this, on the
 * client-side proxy the property org.osgi.remote.intents="SOAP HTTP" is set.
 * However, these intents are *not* set on the original ServiceRegistration on
 * the server-side since on the server side the service object is a local pojo
 * that doesn't get accessed over SOAP and HTTP if it was used from there. This
 * API provides access to these extra properties from the server-side.
 *
 * @param sr A ServiceReference of a published service.
 * @return The map of extra properties.
 */
Map getPublicationProperties(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 services running in a framework to be published for remote consumers and the discovery of services running outside a framework. The Discovery service typically accesses metadata external to the OSGi framework.

When the Discovery service publishes a service over its internal protocol it includes the additional metadata of the service that is passed by the distribution software. This metadata is used for filtering of potential candidates.

5.3.1 Functionality

There are two models for sharing information about distributed services in a system. The 'pull' or 'discovery' model looks for services available in the network and 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 'push' model uses knowledge of each service platform's requirements to push down definitions of matching services.

Distributed OSGi supports the pull model in the optional Discovery Service but allows the 'push' model to be supported as an implementation option, since it does not require any new APIs.

The Discovery service interface defines methods that allow 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 remote services.

The strategy and details of an implementation of the discovery service is left to the implementers. 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.

Distributed services in the sense of this document are described in the `ServiceDescription` class. The content of this data container depends on the available information.

The distribution software on the service provider side passes the information about the service provider to the discovery service. The distribution software calls the discovery method `publish()` to make the service discoverable by other OSGi service platforms as well as other external clients capable of understanding the protocol of the discovery service implementation. The information about the service may only contain those bindings that the distribution software is able to service. In the case of multiple distribution software implementations in the same platform, multiple invocations of `publish()` may occur with different bindings. Figure 9 illustrates how multiple discovery services could be used by the distribution software to publish a service to more than one type of discovery mechanism (such as SLP and UDDI).

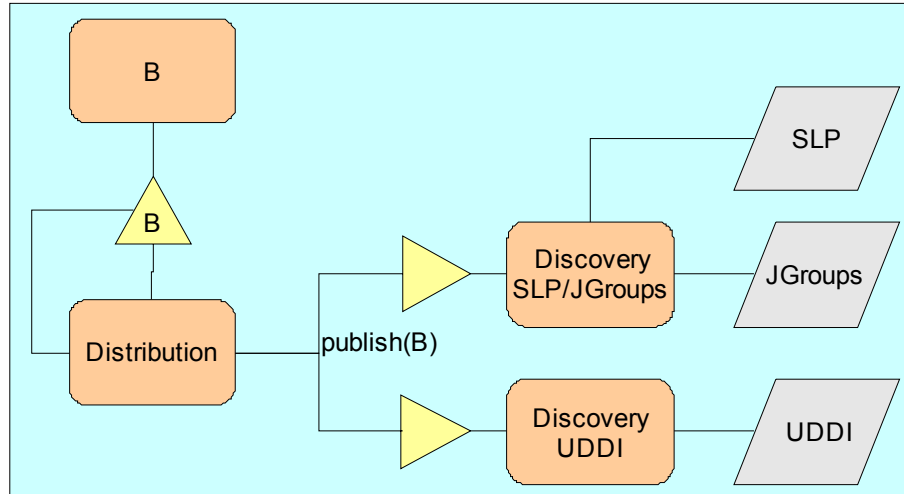


Figure 9: OSGi service published over multiple protocols

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, and web services using SOAP/HTTP(S). Likewise, OSGi services are published using the standard protocol to clients built on other technologies than OSGi.

5.3.2 Discovery using a local file(s)

The discovery mechanism for metadata in a local file or directory was born from the 'I want to connect to a Google Service' use-case. Basically, a mechanism is needed to specify that the details of a service without this information having to access an external discovery service.

The file based discovery service is used in exactly the same way as the External Discovery service, except that it is not based on a real discovery service that lives somewhere else, but populated by information in bundles that are part of the local OSGi installation (it could also be configured using other information, e.g. using the Config Admin Service).

The file based service uses the extender model to check bundles for the existence of a `remote-services.xml` file and if such a file is found it would register discovery information about those services the same way as if it had discovered the service using a remote discovery mechanism. The information registered is exactly the same as the information an external discovery service would hold.

The following is an example of such a file:

```
<?xml version="1.0" encoding="UTF-8"?>
<service-descriptions xmlns="http://www.osgi.org/xmlns/rs/v1.0.0">
  <service>
    <provide interface="com.ionasoftware.pojo.hello.HelloService"/>
    <property name="org.osgi.remote.intents">SOAP HTTP</property>
    <property name="org.osgi.remote.configuration.type">pojo</property>
    <property
name="org.osgi.remote.address">http://localhost:9000/hello</property>
  </service>
  <service>
    <provide interface="com.ionasoftware.pojo.hello.GreeterService"/>
    <property name="org.osgi.remote.intents">SOAP HTTP</property>
    <property name="org.osgi.remote.configuration.type">pojo</property>
  </service>
</service-descriptions>
```

```
<property
name="org.osgi.remote.address">http://localhost:9005/greeter</property>
</service>
</service-descriptions>
```

Note that this XML file is only an example to illustrate the type of format that could be used to contain distributed service metadata in a local file. What's important is the service interface name(s) and associated properties.

The solution should use a folder named "OSGI-INF/remote-services" and parse all files in this folder per default (i.e., adopt the Spring model).

The location for the service description folder location and individual files within it (represented using a comma-separated list) can be overridden by a specific header named "Remote-Service". Multiple headers are allowed. Use the Spring approach for the path specification in the header (e.g. wildcard support, etc.)

If the header contains a PID (bundle symbolic name + version) then it would also be possible to obtain the metadata from Config Admin to configure the service descriptions (but not the location).

5.3.3 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.

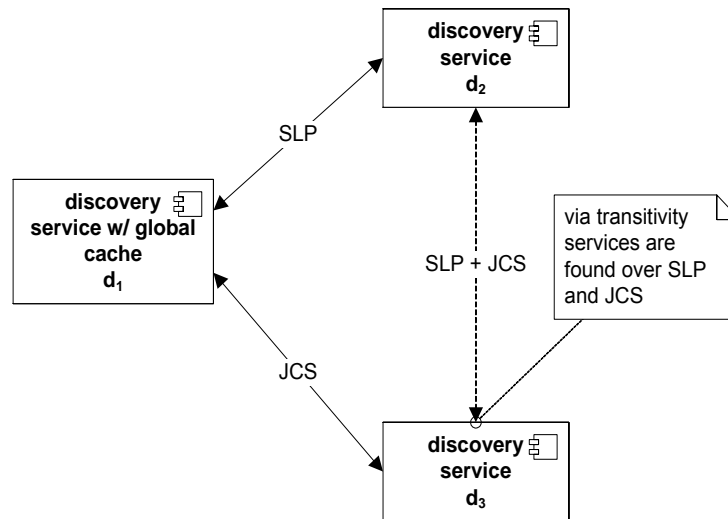


Figure 10: multiple discoveries over different protocols

The possible interaction of multiple OSGi service platforms over the discovery mechanism is shown in Figure 10. 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.

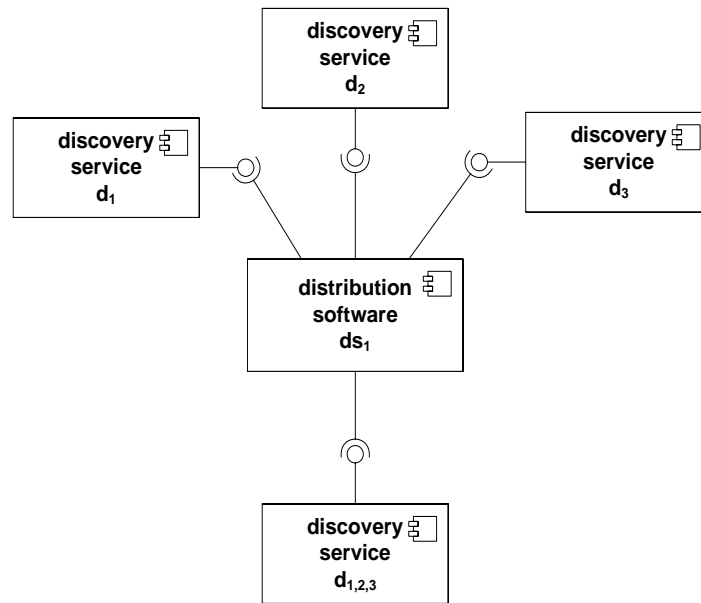


Figure 11: possible discovery service implementation

As shown in Figure 11, 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.

5.3.4 Useful Discovery Service Properties

The following properties are defined for the Discovery service as hints to the implementation based on common use case requirements. An implementation is not required to honor an optional property. It would be helpful to the user of the discovery service to be able to cache service descriptions and auto publish them, for example, but these capabilities are not mandated.

- `org.osgi.discovery.strategy.cache` - [global|local]: the value 'global' instructs the service implementation to try and cache all information about services it has discovered so far, local as well as remote. The feasibility of this strategy depends on the implemented discovery protocol. If the implementation is not able to get notified for services availability changes then it may only rely on the (stale) information it has already received. The value 'Local' instructs the service implementation to cache all service information it has seen in a local storage. This is mainly for faster retrieval.
- `org.osgi.discovery` - [none|auto-publish]: The default value 'auto-publish' instructs the Discovery service implementation to immediately actively push the service information to the network. The value 'none' would make the Discovery service wait for requests from the network before the service information is published.

5.3.5 Interface description

5.3.5.1 Java interface description

Discovery service

```

public interface Discovery {
    final String ORG_OSGI_DISCOVERY = "org.osgi.discovery";
    final String ORG_OSGI_DISCOVERY_NONE = "none";
    final String ORG_OSGI_DISCOVERY_AUTO_PUBLISH = "auto-publish";
}
  
```

```
/**
 * Add a ServiceListener
 *
 * @param listener
 */
void addServiceListener(ServiceListener listener);

/**
 * Remove a ServiceListener
 *
 * @param listener
 */
void removeServiceListener(ServiceListener listener);

/**
 * Compares the given ServiceDescription with the information in the local or global cache
 (depending on the
 * cache strategy set in the properties of the Discovery implementation).<b>
 * The ServiceDescription is matched using the Comparable interface.
 * @param serviceDescription
 * @return true if a service matching the given serviceDescription is found in the local
 cache
 * @throws IllegalArgumentException if serviceDescription is null or incomplete
 */
boolean isCached(ServiceDescription serviceDescription);

/**
 * Returns an array of all ServiceDescription objects currently known to the Discovery
 implementation.
 * @return An array of ServiceDescription objects. An empty array is returned if no service
 description
 * was found.
 */
ServiceDescription[] getCachedServiceDescriptions();

/**
 * Find a service based on the provided service description. The match is performed through
 the Comparable interface
 * implementation of ServiceDescription.
 * @param serviceDescription ServiceDescription of the service to locate
 * @return Collection of ServiceDescription objects matching the service that was found to
 satisfy the find criteria.
 * The Collection may be empty if none was found
 * @throws IllegalArgumentException if serviceDescription is null or incomplete
 */
Collection findService(ServiceDescription serviceDescription);

/**
 * Find a service based on the provided service description and filter.
 * Discovery implementations might choose to not support this method if the discovery
 protocol doesn't support filtering.
 * The match is performed through the Comparable interface implementation of
 ServiceDescription.
 * @param serviceDescription ServiceDescription of the service to locate.
 * @param filter an LDAP filter which the service has to satisfy.
 * @return Collection of ServiceDescription objects matching the service that was found to
 satisfy the find criteria.
 * The Collection may be empty if none was found
 * @throws IllegalArgumentException if serviceDescription is null or incomplete
 * @throws UnsupportedOperationException if method is not supported by the implementation
 */
Collection findService(ServiceDescription serviceDescription, String filter);

/**
 * Asynchronous interface to initiate the search for an suitable service based on the
 provided ServiceDescription.
 * The ServiceDescription is matched using the Comparable interface.
 * @param serviceDescription ServiceDescription of the service to locate
```

```
operation
    * @param callback Listener object to notify about the asynchronous response of the find
    * @throws IllegalArgumentException if the serviceDescription is null or incomplete
    */
    void findService(ServiceDescription serviceDescription, ServiceListener callback);

    /**
     * Asynchronous interface to initiate the search for a suitable service based on the
     * provided ServiceDescription and filter.
     * Discovery implementations might choose to not support this method if the discovery
     * protocol doesn't support filtering.
     * The ServiceDescription is matched using the Comparable interface.
     * @param serviceDescription ServiceDescription of the service to locate
     * @param filter an LDAP filter which the service has to satisfy.
     * @param callback Listener object to notify about the asynchronous response of the find
     * operation
     * @throws IllegalArgumentException if the serviceDescription is null or incomplete
     * @throws UnsupportedOperationException if method is not supported by the implementation
     */
    void findService(ServiceDescription serviceDescription, String filter, ServiceListener
callback);

    /**
     * Publish the provided service. The information is published by the Discovery
     * implementation.<b>
     * If the property osgi.discovery = auto-publish, the Discovery implementation actively
     * pushes the
     * information about the service to the network. Otherwise, it is just available upon
     * request from other
     * Discovery implementations.
     * The ServiceDescription is matched using the Comparable interface.
     * @param serviceDescription ServiceDescription of the service to publish
     * @return true if the service was successfully published.
     * @throws IllegalArgumentException if serviceDescription is null or incomplete
     */
    boolean publish(ServiceDescription serviceDescription);

    /**
     * Publish the provided service. The information is published by the Discovery
     * implementation.<b>
     * If the parameter autopublish=true, the Discovery implementation actively pushes the
     * information about the service to the network. Otherwise, it is just available upon
     * request from other
     * Discovery implementations.
     * The ServiceDescription is matched using the Comparable interface.
     * @param serviceDescription ServiceDescription of the service to publish
     * @param autopublish if true, service information is actively pushed to the network for
     * discovery
     * @return true if the service was successfully published.
     * @throws IllegalArgumentException if serviceDescription is null or incomplete
     */
    boolean publish(ServiceDescription serviceDescription, boolean autopublish);

    /**
     * Make the given service un-discoverable. The previous publish request for a service is
     * undone. The service
     * information is removed from the local or global cache.
     * The ServiceDescription is matched using the Comparable interface.
     * @param serviceDescription ServiceDescription of the service to unpublish
     * @throws IllegalArgumentException if serviceDescription is null or incomplete
     */
    void unpublish(ServiceDescription serviceDescription);
}


```

ServiceListener

```
public interface ServiceListener {
    /**
     * Callback indicating that the specified service was discovered and is known to the calling
     * Discovery implementation.
     * @param serviceDescription
     */
    void serviceAvailable(ServiceDescription serviceDescription);

    /**
     * Callback indicating a change in the service description of a previously discovered service.
     * @param oldDescription previous service description
     * @param newDescription new service description
     */
    void serviceModified(ServiceDescription oldDescription, ServiceDescription newDescription);

    /**
     * Callback indicating that the specified service is no longer available/
     * @param serviceDescription ServiceDescription of the service that is no longer available
     */
    void serviceUnavailable(ServiceDescription serviceDescription);
}
```

ServiceDescription

```
public interface ServiceDescription extends Comparator {
    /**
     * @return The service interface name
     */
    String getInterfaceName();

    /**
     * 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 <code>java.util.Collection</code> of the property names
     * available in the ServiceDescription
     */
    Collection keys();

    /**
     * @return Returns all properties of the interface as a
     * <code>java.util.Map</code>.
     */
    Map getProperties();
}
```

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].

5.5 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 types 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 DSW are deployed, a service can be optionally configured using additional SCA metadata (see section X [tbs]). Other

metadata forms are also permitted (standard or proprietary), through an extensibility mechanism but their integration into OSGi is not defined.

In the following sections each type of metadata will be described as well as their interdependencies. Intents can be part of the bundle manifest.mf, an xml file within a bundle, or service configuration data

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 DSW interface, and some restrictions on the Java interface are therefore necessary to ensure compatibility across multiple DSW types (see Section 5.7).

5.5.2 Properties

Property – a property of a service is used to describe it 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 code 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

- `org.osgi.remote.publish` – ["*" | comma-separated interface name list]: A distribution software implementation may create and register a binding for a service, 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. If the property value 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.
- `org.osgi.remote.intents` – optional list of intents defined by the component designer and changeable by the deployer
- `org.osgi.remote.configuration.type` – identifies the metadata type of additional metadata, if any, that was provided with the service provider or consumer, e.g. "SCA"

Because distributed OSGi is designed for compatibility with existing distributed software systems, only the `org.osgi.remote.publish` property is required. When a DSW encounters this property it knows that more metadata is required to deploy the service, and uses one of the discovery service mechanisms to obtain it.

The `intents` property optionally defines the QoS capabilities that a published service provides, and allows a service requester to filter remote services according to its desired QoS capabilities.

The `configuration.type` optionally defines portable metadata to address the requirement for consistence across multiple DSW types, for the case in which multiple DSW types are involved.

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 version="1.0" encoding="UTF-8"?>
<component name="OrderBeerService">
  <implementation .../>
  <service>
    <provide interface="org.beer.OrderBeerService" >
      <property name="org.osgi.remote.publish">*</property>
      <property name="org.osgi.remote.intents">confidentiality</property>
    </provide />
  </service>
</component>
```

The example illustrates the `org.osgi.remote.publish` and `org.osgi.remote.intents` properties specified for the `OrderBeerService`, which are associated with its interface. The `confidentiality` intent specifies the capability of the service to support encryption, such as through HTTP or IOP/SSL.

5.5.3 Intents

An *intent* is an abstraction of a distributed computing capability that can be used to provision and select services. An intent describes one or more requirements of a service provider and consumer on a service published by the distribution software in use.

An intent is a high-level, generic statement of 'what' a consumer may require of 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 an intent, and describes how to define additional intents.

A service requester can use an intent to help select 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 DSW and its (default) configuration. By default, no intents are attached to a 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 are listed in a service property named `org.osgi.remote.intents`.

5.5.3.1 Example of using Intents

The example below shows how the 'confidentiality' and the 'reliable' intents can be added to an example `OrderBeerService`. This might be desirable in order to prevent people snooping on messages and in order to ensure that orders are not lost. These would typically be implemented using encryption and a reliable transport, respectively.

The following example shows how a `OrderBeerService` requires a `BeerWarehouseService` which is also available over a reliable transport, expressed using the `reliable` intent.

Example XML file used by Declarative Services:

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="OrderBeerService">
  <implementation .../>
  <service>
    <provide interface="org.beer.OrderBeerService" >
      <property name="org.osgi.remote.publish">*</property>
      <property name="org.osgi.remote.intents">
        confidentiality
        reliable
      </property>
    </provide />
  </service>
  <reference .../>
  <reference name="BeerWarehouseService"
    interface="org.beer.BeerWarehouseService" />
</component>
```

The example illustrates that the service reference to `BeerWarehouseService` filters the remote service using the `reliable` intent.

5.5.3.2 Defining Intents

An intent is string with an associated abstract meaning (see Section 5.5.3.4 for the list of SCA defined intents). 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 custom intents using the mechanism defined in Section 5.5.3.4.

5.5.3.3 OSGi-defined Intents

OSGi defines one intent, '`passByReference`', to allow services to specify whether they require by-reference call semantics.

- `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 this intent is not used, the pass-by semantics are determined by the DSW.

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'). Below are examples from the SCA Policy Framework specification ([add ref](#)). Note, the '.' is used to define 'qualified intents' which are described in more detail in section 5.5.3.5.

- authentication
 - authentication.message
 - authentication.transport
- confidentiality
 - confidentiality.message
 - confidentiality.transport
- integrity
 - integrity.message
 - integrity.transport
- reliability
- ordered

SCA also defines a schema for adding new intents. A Distribution Software may choose to support this schema as a mechanism for adding new intent definitions.

The SCA pseudo-schema for intent definition is as follows:

```
<intent name="NCName"
  constrains="listOfQNames"
  requires="listOfQNames"? >
  <description>
    <!-- description of the intent -->
  </description>
</intent>
```

Where

- `@constrains`: specifies the construct that this intent is meant to apply to.
- `@requires`: defines the set of all intents that the referring intent requires. This allows intents to be composed out of other intents.

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:

```
<intent name="communicationProtection"
  constrains="binding"
  requires="confidentiality integrity">
  <description>
    Ensure that communications cannot be seen or tampered with by
    unauthorized personnel.
  </description>
</intent>
```

5.5.3.5 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' to 'confidentiality.message.body'.

Please look at [6] for further details on qualified intents.

5.5.3.6 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 service has originally provided a qualified intent, then Distribution Software should list also all 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 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 Distribution Software implemented 'confidentiality' at the message level. So clients looking directly for qualified intents can also be served.

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 property is a URL relative to the JAR of the bundle (i.e. it's a resource in the bundle). Any sub-properties such as the protocol type property can be represented either as a singleton or as an array.

In general, metadata in a configuration type can be used to create a machine-readable description of a remotd 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 higher than matches found using additional metadata. It should also be possible to rank services using a comparator.

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

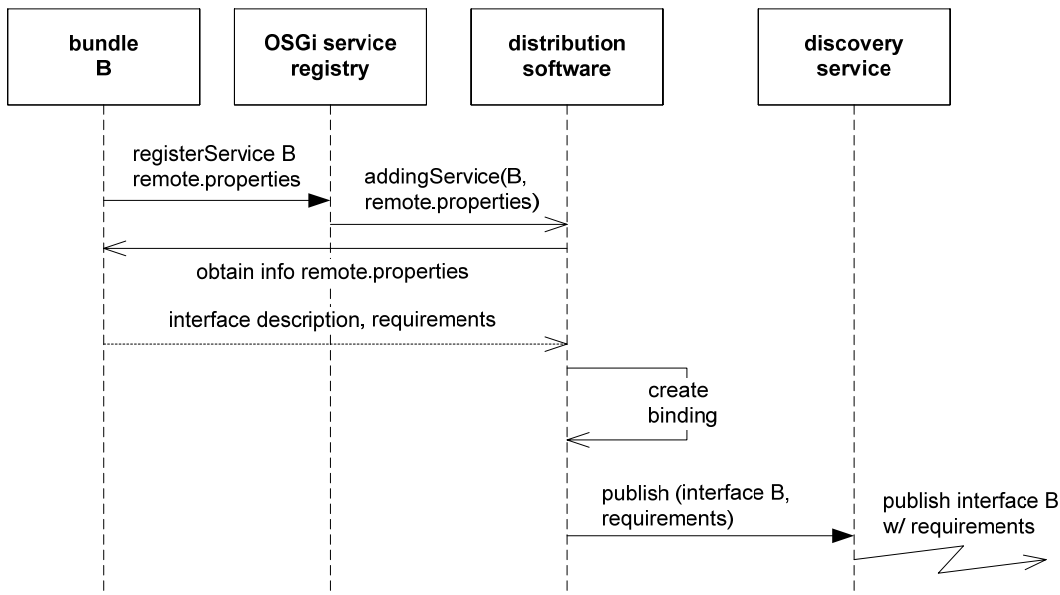


Figure 12: Server side service registration

How the service B is registered in the OSGi framework and then made available for remote access is shown in Figure 12. The important part in the picture is that the interface B is augmented with additional metadata or service contract information, which enables the distribution software to pick the appropriate protocol for the service binding and publish the service availability and QoS parameters using the Discovery service. The service metadata is obtained from the providing bundle and service properties that are part of the service registration. The details about the metadata are explained further in section 5.5.

5.6.1.2 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.

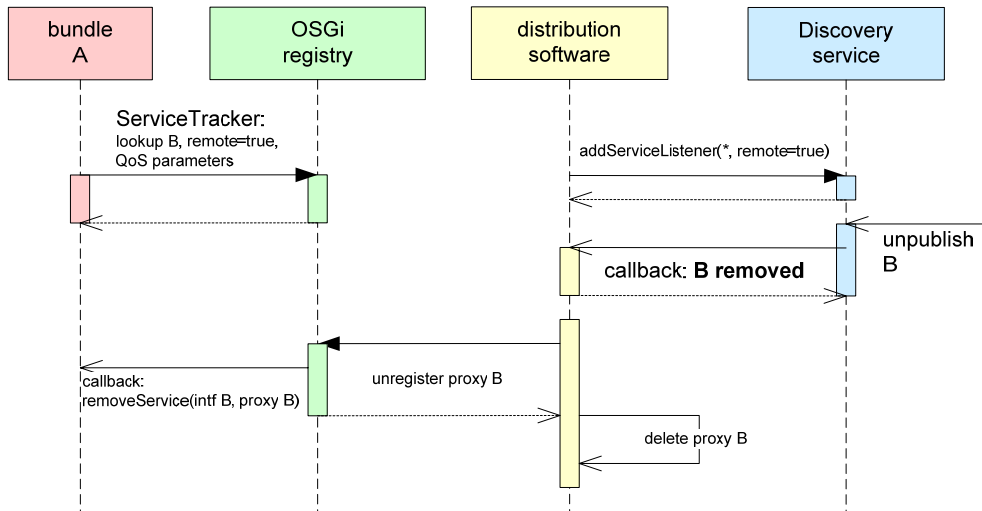


Figure 13: Unregistration of a service interface

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. It is the Distribution Software's responsibility to ensure any proxies associated with the metadata are unregistered.

5.6.2 Interactions on the service consumer side

5.6.2.1 Lookup for a remote Service

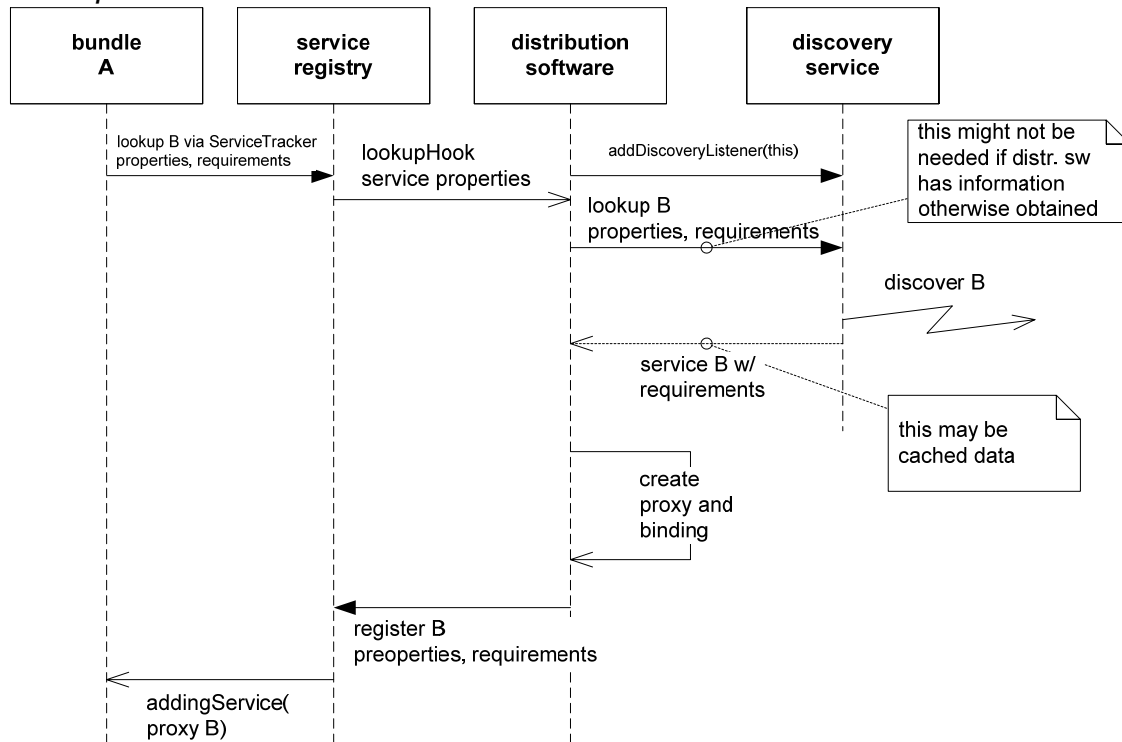


Figure 14: Client side service lookup

In Figure 14 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 returned to the service consumer via the callback. 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: `org.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.

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.

5.6.3 Interactions with Non-OSGi service providers and consumers

[tbs]

5.6.4 Lifecycle dynamics

Activation of distribution software - When a distribution software becomes installed and activated, it should check for:

- any services that are already in the OSGi Service Registry and need to be exposed remotely
- any active bundle which contains unresolved references to remote services. Metadata describing the service may also already exist, being provided manually and not via Discovery Service, and should be used appropriately.

Deactivation of distribution software - Discovery services should propagate any resulting changes of service availability. The distribution software should unregister any proxies it has registered in the OSGi registry.

Activation of a discovery service – Distribution software should check for any active bundle having unresolved references to remote services and try to resolve them with the help of the new Discovery Service.

Deactivation of a discovery service – no special actions are required. The administrator should be aware that this will mean the framework will not be notified when previously discovered services become unavailable.

Changes of service metadata at runtime -

Changes to defined as well as optional metadata are theoretically possible e.g.

- for defined metadata: value of a property is changed via CAS, intent/policySet was changed/removed
- for optional metadata: new distribution sw providing additional bindings has started

It would make sense to restrict changes on defined metadata if service has already consumers because the consumer might have done his service provider choice based on that metadata (service contract!). For the same reasons could the removal of optional intents be critical (because they got used).

Additionally, new distribution software and/or distribution configuration can be added after the service was registered which can cause the registered service to then be distributed.

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 Best Practices

This section is non-normative.

5.7.1.1 Runtime (Framework)

Controls the lifecycle of services and service dependencies (e.g. DS, Spring). Unresolved packages, class loading issues are indicators for improper configuration by the deployer. Runtime enforces policies defined by the architect and solution designer to guarantee appropriate match up of consumer and producer.

5.7.2 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.

5.7.3 Connector

[tbs]

5.7.4 Caching

[tbs]

5.7.5 Automated Service discovery

[tbs]

5.7.6 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.

5.7.7 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.

5.8 Reference Implementation

5.8.1 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.

5.9 Reference Implementation based on SCA

6 Considered Alternatives

6.1.1 Alternative: using simple properties to define service remoting

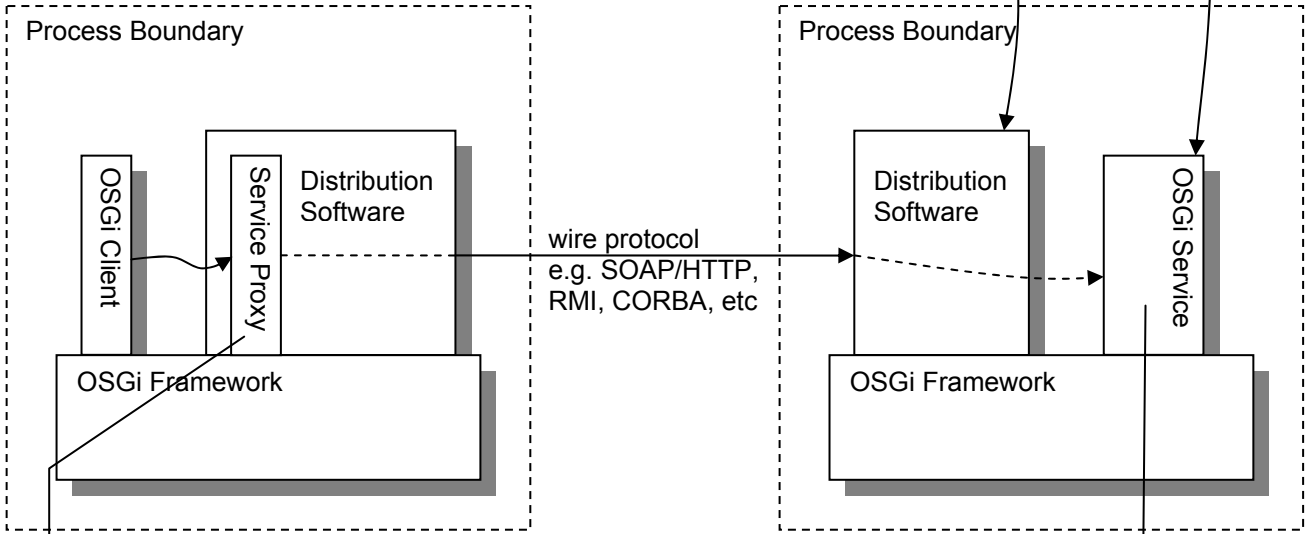
This alternative was not considered viable as the simple properties approach is most likely not expressive enough.

Properties specified by Deployer in configuration for the Service, e.g. in a DS XML file:

```

<component name="example.component">
  <implementation class="com.example.Component"/>
  <property name="remote.profile">SOAP/HTTP</property>
  <service>
    <provide interface="com.example.ServiceInterface"/>
  </service>
</component>

```



Properties (Client-side, read only):

```

remote = true
remote.profile = SOAP/HTTP
remote.url =
  http://server:1234/myService
remote.interface =
  http://server:1234?wsdl

```

Properties (Server-side, read only):

```

remote = false
nothing else mandated

```

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 remotod, 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? It 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.

7 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.

8 Document Support

8.1 References

- [1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
- [2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0
- [3]. OSGi RFP 79: Remote Service Discovery.
<https://www2.osgi.org/members/svn/documents/trunk/rfps/rfp-0079-ServiceDiscovery.doc>
- [4]. OSGi RFP 88: Externalize OSGi Services.
<https://www2.osgi.org/members/svn/documents/trunk/rfps/rfp-0088-ExternalServices.doc>
- [5]. OSGi RFC 126: <https://www.osgi.org/members/svn/documents/trunk/rfcs/rfc0126/rfc-0126-ServiceRegistryHooks.odt>
- [6]. SCA Policy Framework 1.00, <http://www.oasis-opencsa.org/SCA-policy-framework>
- [7]. SCA Web Service Binding Specification 1.00, <http://www.oasis-opencsa.org/sca-bindings>
- [8]. SCA JMS Binding Specification 1.00, <http://www.oasis-opencsa.org/sca-bindings>

8.2 Author's Address

Name	Eric Newcomer
Company	IONA Technologies
Address	200 West Street, Waltham, MA 02451 USA
Voice	+1 781 902 8366
e-mail	eric.newcomer@iona.com

Name	David Bosschaert
Company	IONA Technologies
Address	The IONA Building, Shelbourne Road, Ballsbridge, Dublin 4, Ireland
Voice	+353 1 637 2371
e-mail	davidb@iona.com

Name	Tim Diekmann
Company	TIBCO Software
Address	3303 Hillview Ave, Palo Alto, CA 94034, USA
Voice	+1-650-846-5521
e-mail	tdiekman@tibco.com

8.3 Acronyms and Abbreviations

OASIS	Organization for the Advancement of Structured Information Standards
Open CSA	Open Composite Services Architecture
SCA	Service Component Architecture
WSDL	Web Services Description Language

8.4 End of Document



RFC 124:

A Component Model for OSGi

0.93 Draft

73 Pages

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

Copyright © The OSGi Alliance 2008.

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 Status	3
0.3 Terminology and Document Conventions	3
0.4 Revision History	3
1 Introduction	4
2 Application Domain	5
2.1 Terminology and Abbreviations	5
3 Problem Description	6
4 Requirements	7
5 Solution	8
5.1 Architectural Overview	8
5.2 Module Context Life Cycle and the Extender Bundle	10
5.2.1 Module context creation and destruction	10
5.2.2 Manifest Headers for Managed Bundles	11
5.3 Declaring Module Components	13
5.3.1 Naming Components	14
5.3.2 Instantiating Components	14
5.3.3 Dependencies	15
5.3.4 Component Scopes	26
5.3.5 Lifecycle	27
5.4 Interacting with the Service Registry	28
5.4.1 Exporting a managed component to the Service Registry	28
5.4.2 Defining References to OSGi Services	31
5.4.3 Dealing with service dynamics	35
5.5 Configuration Administration Service Support	38
5.5.1 Property Placeholder Support	38
5.5.2 Managed Services	39
5.5.3 Managed Service Factories	40
5.5.4 Direct access to configuration data	40
5.5.5 Publishing Configuration Admin properties with exported services	41
5.6 APIs	41

5.6.1 ServiceUnavailableException41

5.6.2 ModuleContextListener **Error! Bookmark not defined.**

5.7 'osgi' Schema64

5.8 'osgix' Schema.....71

6 Considered Alternatives72

7 Security Considerations72

8 Document Support72

8.1 References.....72

8.2 Author's Address73

8.3 Acronyms and Abbreviations.....73

8.4 End of Document.....73

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.

Revision	Date	Comments
0.1 draft	Sep 13 th 2007	First draft of this RFC Adrian Colyer, SpringSource, adrian.colyer@springsource.com
0.2 draft	Nov 28th 2007	Second draft of RFC2, incorporating design material from Spring Dynamic Modules 1.0 rc1
0.5 draft	Dec 21st 2007	Added first version of section 5.3. Defined mechanism for accessing ServiceReference objects for a collection.
0.8 draft	March 25th 2008	Addressed comments from January 2008 EEG meeting, completed section 5.3, added ModuleContextListener type.
0.9 draft	May 16th 2008	Addressed comments from John Wells and Alexandre Alves posted to EEG mailing list

Revision	Date	Comments
0.9.1 draft	June 17th 2008	Addressed bugs <ul style="list-style-type: none">702: don't proxy service references703 management of service references within a collection704 Class of a lazy init component is not reference until the component is about to be instantiated
0.9.2 draft	July 3rd 2008	Addressed bugs <ul style="list-style-type: none">668: signature of destroy-method callback672: use "mandatory" and "optional" instead of 1..N etc.701: use 'Map' instead of 'Dictionary'667 remove 'legacy' spring constructs695 define API for accessing components and meta-data
0.9.3 draft		Addressed bugs <ul style="list-style-type: none">699 specification of type conversionsCompleted section 5.2.3, Module Context Events

1 Introduction

The OSGi Alliance needs an amended process for the creation and approval of press releases. This document will describe the process.

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. JavaServlets: 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 **Error! Reference source not found**.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 instantiate and configure the required components when the bundle is started. A bundle with such information present is known as a *managed bundle*.

An extender bundle is responsible for observing the life cycle of such bundles. When a bundle is started, the extender creates a *module context*¹ for that bundle 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.

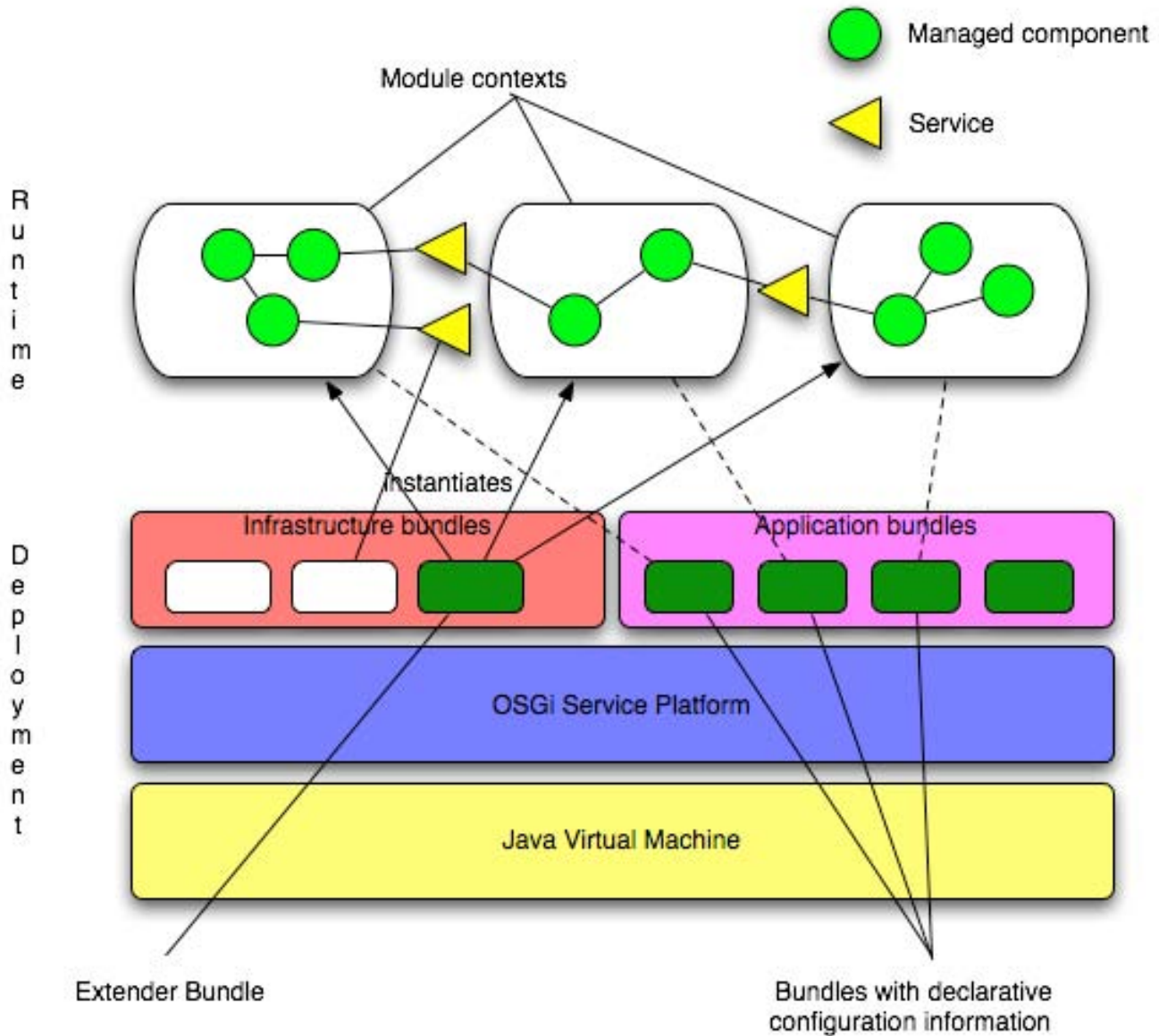
The declarative configuration for a bundle may also specify that certain of the bundle's managed components are to be exported as services in the OSGi service registry. In addition, it is possible to declare that a bundle component depends on a service or set of services obtained via the service registry, 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.

¹ It is tempting to call this a “bundle context”, but that could cause confusion with the BundleContext interface. In Spring this concept is known as an “application context”.

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.

Figure 1 Solution Overview



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

Implementations of this RFC must provide an extender bundle with symbolic name `org.osgi.service.context.extender`. 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 starting 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 STARTED state before its module context has been created. It is possible to force synchronous creation of module contexts on a bundle-by-bundle basis. See Section 5.2.2, "Manifest headers" for information on how to specify this behavior.

If module context creation fails for any reason then logged 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.

A client interested in the creation of module contexts (in either the success or failure cases) may register a service of type `org.osgi.module.context.ModuleContextListener` in the service registry in order to receive module events.

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. 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 configured to not wait for dependencies, a bundle with unsatisfied mandatory dependencies will be stopped, leaving the bundle in the RESOLVED state.

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.

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 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).

5.2.2 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 META-INF/module-context with one or more files in that folder with a '.xml' extension.
- META-INF/MANIFEST.MF contains a manifest header Module-Context.

In addition, if the optional ModuleContextExtender-Version header is declared in the bundle manifest, then the extender will only recognize bundles where the specified version constraints are satisfied by the version of the extender bundle (Bundle-Version). The value of the ModuleContextExtender-Version header must follow the syntax for a version range as specified in section 3.2.5 of the OSGi Service Platform Core Specification.

In the absence of the Module-Context header the extender expects every ".xml" file in the META-INF/module-context folder to be a valid module context configuration file, and all directives (see below) take on their default values. A single module context is constructed from this set of files.

The Module-Context 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 Module-Context header defines at least one configuration file location, any files in META-INF/module-context are ignored unless directly referenced from the Module-Context header.

The syntax for the Module-Context header value is:

```
Module-Context-Value ::= context ( ',' context ) *  
context ::= path ( ';' path ) * ( ';' directive ) *
```

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.

For example, the manifest entry:

```
Module-Context: 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.

The wildcard "*" can be used to match zero or more path characters. A "*" used on its own, as in:

```
Module-Context: *
```

Matches all XML files in META-INF/module-context/*.xml (i.e. the default behavior).

A number of directives are available for use with the Module-Context header. These directives are:

- `create-asynchronously (false|true)`
controls whether the module context is created asynchronously (the default), or synchronously.

For example: `Module-Context: *;create-asynchronously=false`

Creates a module context synchronously, using all of the "*.xml" files contained in the META-INF/module-context folder.

And: `Module-Context: config/account-data-context.xml;create-asynchronously:=false`

Creates a module context synchronously using the `config/account-data-context.xml` configuration file. Care must be taken when specifying synchronous context creation as the module context will be created on the OSGi event thread, blocking further event delivery until the context is fully initialized. If an error occurs during the synchronous creation of the module context then a `FrameworkEvent.ERROR` event is raised. The bundle will still proceed to the ACTIVE state.

- `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: `Module-Context: config/osgi-*.xml;wait-for-dependencies:=false`

Creates a module context using all the files matching "osgi-*.xml" in the config directory. Context creation will begin immediately even if dependencies are not satisfied. This essentially means that mandatory service references are treated as though they were optional - clients 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.

- `timeout (300)`

the time to wait (in seconds) for mandatory dependencies to be satisfied before giving up and failing module context creation. This setting is ignored if `wait-for-dependencies:=false` is specified. The default is 5 minutes (300 seconds).

For example: `Module-Context: *;timeout:=60`

Creates an application context that waits up to 1 minute (60 seconds) for its mandatory dependencies to appear.

If there is no `Module-Context` manifest entry, or no value is specified for a given directive in that entry, then the directive takes on its default value.

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.module.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.module.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/module/context/CREATING` – the extender has started to create a module context
- `org/osgi/module/context/CREATED` – a module context has been successfully created
- `org/osgi/module/context/DESTROYING` – the extender is destroying a module context
- `org/osgi/module/context/DESTROYED` – a module context has been destroyed
- `org/osgi/module/context/WAITING` – creation of a module context is waiting on the availability of a mandatory service

- `org/osgi/module/context/FAILURE` – creation of a module context has failed

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_VERSION` (Version) the version of the bundle for which the context is being created / destroyed
- `TIMESTAMP` (Long) the time when the event occurred

In addition for a `FAILURE` event the `EXCEPTION` property contains a Throwable detailing the failure cause. For a `WAITING` event, the `SERVICE_CLASS` (String) property details the type of the service that the context is waiting on, and the `SERVICE_FILTER` (String) property details the filter (if any).

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 or creation times out.

5.3 Declaring Module Components

A module context configuration file contains component definitions using XML declarations from the `osgi` namespace (see section 5.7). The module context container manages the lifecycle of these components. The basic structure of a configuration file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://www.osgi.org/schema/comp"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.osgi.org/schema/comp http://www.osgi.org/schema/comp/osgi-comp-r5.xsd">

  <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>
```

A module container manages one or more *components*. These components are created using the configuration metadata 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 one or more ids (also called identifiers, or names; these terms refer to the same thing). These ids must be unique within the module the component is hosted in. A component will almost always have only one id, but if a component has more than one id, the extra ones can essentially be considered aliases.

The 'id' or 'name' attributes are used to specify the component identifier(s). The 'id' attribute allows you to specify exactly one id, and as it is a real XML element ID attribute, the XML parser is able to do some extra validation when other elements reference the id; as such, it is the preferred way to specify a component id. However, the XML specification does limit the characters which are legal in XML IDs. This is usually not a constraint, but if you have a need to use one of these special XML characters, or want to introduce other aliases to the component, you may also or instead specify one or more component ids, separated by a comma (,), semicolon (;), or whitespace in the 'name' attribute.

Please note that you are not required to supply a name for a component. If no name is supplied explicitly, the container will generate a unique name for that component. The motivations for not supplying a name for a component will be discussed later (one use case is inner components).

5.3.1.1 Aliasing components

In a component definition itself, you may supply more than one name for the component, by using a combination of up to one name specified via the id attribute, and any number of other names via the name attribute. All these names can be considered equivalent aliases to the same component, and are useful for some situations, such as allowing each component used in an application to refer to a common dependency using a component name that is specific to that component itself.

5.3.2 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).

5.3.2.1 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.

5.3.2.2 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.

5.3.2.3 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 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.

```
<!-- the factory component, which contains a method called createInstance() -->
<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"/>
```

5.3.3 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.

5.3.3.1 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.

Constructor Injection

Constructor-based DI is effected by invoking a constructor with a number of arguments, each representing a dependency. Additionally, calling a static 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 static factory method similarly. Find below an example of a class that could only be dependency injected using constructor injection. Notice that there is nothing *special* about this class.

```
public class SimpleMovieLister {
    // the SimpleMovieLister has a dependency on a MovieFinder
    private MovieFinder movieFinder;
    // 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 argument resolution matching occurs using the argument's type. If there is no potential for ambiguity in the constructor arguments of a component definition, then the order in which the constructor arguments are defined in a component definition is the order in which those arguments will be supplied to the appropriate constructor when it is being instantiated. Consider the following class:

```
package x.y;
public class Foo {
    public Foo(Bar bar, Baz baz) {
        // ...
    }
}
```

There is no potential for ambiguity here (assuming of course that Bar and Baz classes are not related in an inheritance hierarchy). Thus the following configuration will work just fine, and you do not need to specify the constructor argument indexes and / or types explicitly.

```
<components>
  <component name="foo" class="x.y.Foo">
    <constructor-arg>
      <component class="x.y.Bar"/>
    </constructor-arg>
    <constructor-arg>
      <component class="x.y.Baz"/>
    </constructor-arg>
  </component>
</components>
```

When another component is referenced, the type is known, and matching can occur (as was the case with the preceding example). When a simple type is used, such as specifying value="true", the container cannot determine the type of the value, and so cannot match by type without help. Consider the following class:

```
package examples;
public class ExampleComponent {
    // No. of years to the calculate the Ultimate Answer
    private int years;
    // The Answer to Life, the Universe, and Everything
    private String ultimateAnswer;
    public ExampleBean(int years, String ultimateAnswer) {
        this.years = years;
        this.ultimateAnswer = ultimateAnswer;
    }
}
```

The above scenario *can* use type matching with simple types by explicitly specifying the type of the constructor argument using the 'type' attribute. For example:

```
<component id="exampleComponent" class="examples.ExampleComponent">
    <constructor-arg type="int" value="7500000"/>
    <constructor-arg type="java.lang.String" value="42"/>
</component>
```

Constructor arguments can 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>
```

As well as solving the ambiguity problem of multiple simple values, specifying an index also solves the problem of ambiguity where a constructor may have two arguments of the same type. Note that the *index is 0 based*.

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 properties could be specified via constructor injection, and optional properties via setter injection. Properties are assumed to follow JavaBeans conventions.

Here is an example:

```
<component id="exampleComponent" class="examples.ExampleComponent">
    <!-- setter injection using the nested <ref/> element -->
    <property name="componentOne" ref="anotherComponent"/>
    <!-- setter injection using the neater 'ref' attribute -->
    <property name="componentTwo" ref="yetAnotherComponent"/>
    <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.

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>
```

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:

```
<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).

```
<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:

```
<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.

Additionally, if the component being referred to is in the same XML unit, and the component name is the component *id*, the 'local' attribute may be used, which allows the XML parser itself to validate the component *id* even earlier, at XML document parse time.

```
<property name="targetName">
  <!-- a component with an id of 'theTargetComponent' must exist; otherwise an XML
    exception will be thrown -->
  <idref local="theTargetComponent"/>
</property>
```

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.

```
<ref component="someComponent"/>
```

Specifying the target component by using the local attribute leverages the ability of the XML parser to validate XML id references within the same file. The value of the local attribute must be the same as the id attribute of the target component. The XML parser will issue an error if no matching element is found in the same file. As such, using the local variant is the best choice (in order to know about errors as early as possible) if the target component is in the same XML file.

```
<ref local="someComponent"/>
```

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, and it is best not to even specify any id or name value because the id or name value simply will be ignored by the container.

```
<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' flag and any 'id' or 'name' attribute are effectively ignored. Inner components are *always* anonymous.. Please also note that it is *not* possible to inject inner components into collaborating components other than the enclosing component.

Collections

The `<list/>`, `<set/>`, `<map/>`, and `<props/>` elements allow properties and arguments of the Java Collection type List, Set, Map, and Properties, respectively, to be defined and set.

```
<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 bean="myDataSource" />
    </list>
  </property>
  <!-- results in a setSomeMap(java.util.Map) call -->
  <property name="someMap">
    <map>
      <entry>
```

Draft

```
        <key>
          <value>an entry</value>
        </key>
        <value>just some string</value>
      </entry>
    <entry>
      <key>
        <value>a ref</value>
      </key>
      <ref bean="myDataSource" />
    </entry>
  </map>
</property>
<!-- results in a setSomeSet(java.util.Set) call -->
<property name="someSet">
  <set>
    <value>just some string</value>
    <ref bean="myDataSource" />
  </set>
</property>
</bean>
```

Note that the value of a map key or value, or a set value, can also again be any of the following elements:

- component
- ref
- idref
- list
- set
- map
- props
- value
- null

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). That is, it is possible to declare a Collection type such that it can only contain String elements (for example). 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.

```
public class Foo {

    private Map<String, Float> accounts;

    public void setAccounts(Map<String, Float> accounts) {
        this.accounts = accounts;
    }
}

<components>
  <component id="foo" class="x.y.Foo">
    <property name="accounts">
      <map>
        <entry key="one" value="9.99"/>
      </map>
    </property>
  </component>
</components>
```

Draft

```
        <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 supported on Java 5 or higher runtime environments.

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 primitive types (permissible values for booleans are "yes", "no", "true", "false", "on", "off") and their object equivalents (e.g. Float, float).
- String to class
- String to java.io.File
- String to Locale
- String to Pattern (only support on JDK 1.5 or later)
- String to URL
- String to java.lang.Properties (String must follow the format describe in the JavaDoc for Properties)
- Collection to Collection (converts any source collection type to a given target Collection type)
- String to Date (uses the default pattern yyyy-MM-dd)

Additional converters may be defined by implementing the `org.osgi.module.context.convert.Converter` interface.

```
Interface Converter {

    public Class[] getSourceClasses();

    public Class[] getTargetClasses();

    public Object convert(Object source, Class targetClass) throws Exception;

}
```

A bundle can declare a type converter for use by any module context container instantiated for that bundle by defining it with the `Module-Context-Type-Converter` header in `MANIFEST.MF`.

`Module-Context-Type-Converter: fully-qualified-type-name;from="fq1,fq2,fq3";to="fq1,fq2,fq3"`

For example:

```
Module-Context-Type-Converter:
com.xyz.converters.UserIdConverter;from="java.lang.String";to="com.xyz.user.UserId"
```

Draft

Any type converters locally registered in this way take precedence over the default type converters in the case that both a default type converter and a locally-registered type converter are able to perform the same conversion.

In addition to this bundle-specific declaration of type converters, it is also possible for a bundle to globally register type converters to be used by any module context container. A type converter is globally exported by adding the directive "scope:=platform" to the converter declaration.

For example the declaration:

```
Module-Context-Type-Converter: org.local.converter.MyConverter;from="a.b.C";to="d.e.F",  
org.global.converter.GlobalConverter;from="g.h.I";to="j.k.L";scope:=platform
```

makes MyConverter available only within the module context container associated with the bundle, but makes GlobalConverter available for use by all.

A module context container must make available for type conversion all *type-compatible* global converters declared in bundles that have been successfully resolved at the point in time the module context container is instantiated (when its corresponding bundle is started).

Given a bundle C declaring a type converter in its manifest, and a bundle B for which a module context is to be created, then the type converter in C is type-compatible with B iff:

- C is wired to the same exporter of the Converter type as the extender bundle responsible for creating the module context
- B and C are both wired to the same exporter of the Converter type (or B does not import the Converter type)
- For all type names in the "from" attribute of the converter declaration, B and C are both wired to the same exporter of those types
- For all type names in the "to" attribute of the converter declaration, B and C are both wired to the same exporter of those types

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/></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/></property>  
</component>
```

The above configuration is equivalent to the following Java code: `exampleComponent.setEmail(null)`.

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:

Draft

```
<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 bean="myComponent">
</property>
```

```
<constructor-arg>
  <ref bean="myComponent">
</constructor-arg>
```

... are equivalent to:

```
<property name="myProperty" ref="myComponent"/>
```

```
<constructor-arg ref="myComponent"/>
```

Note however that the shortcut form is equivalent to a `<ref bean="xxx">` element; there is no shortcut for `<ref local="xxx">`. To enforce a strict local reference, you must use the long form.

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"/>
```

Again, the shortcut form is equivalent to a `<ref bean="xxx">` element; there is no shortcut for `<ref local="xxx">`.

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 bean is constructed, or a `NullPointerException` will be thrown.

5.3.3.2 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.

```
<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 list of component names as the value of the `'depends-on'` attribute, with commas, whitespace and semicolons all valid delimiters, like so:

```
<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 first prior to the relevant component itself being destroyed. This thus allows you to control shutdown order too.

5.3.3.3 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:

```
<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:

```
<components default-lazy-init="true">
  <!-- no components will be pre-instantiated... -->
</components>
```

(Note to reviewers, I've elected to exclude Spring autowiring support from this version of the specification – please object if you think it should be included).

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.

5.3.4 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:

Scope Name	Description
singleton	scopes a single component definition to a single object instance per module context
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.

```
<component id="accountService" class="com.foo.DefaultAccountService" scope="singleton"/>
```

The non-singleton, prototype scope of component deployment results in the creation of a new component instance every time a request for that specific component is made. 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.

```
<component id="accountService" class="com.foo.DefaultAccountService" scope="prototype"/>
```

See section 5.4.1 for a discussion of the bundle scope, which may be used when exporting a component as an OSGi service.

Implementations of this RFC are free to define additional scope values beyond those defined here.

5.3.5 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.

```
<component id="exampleInitComponent"
    class="examples.ExampleComponent" init-method="init"/>
```

```
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>
```

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 either no arguments or an `int` reason code. NOTE: open discussion point. A component is only destroyed when its enclosing module context is destroyed. A module context is only destroyed when a bundle is stopping. So the only reason code we could ever give at the moment is "BUNDLE_STOPPING". It would be very nice to be able to support "BUNDLE_UPDATING" and "BUNDLE_REFRESHING" as additional codes as well, but this would require a change to the `BundleEvent` so that we could distinguish. Without such a change, is the reason code version really worth specifying?

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. 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 `org.osgi.service.context.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>
```

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. When a service importing bundle is stopped, the component instance associated with it is disposed. To declare a component with bundle scope simply use the scope attribute of the component element:

```
<service ref="compToBeExported" interface="com.xyz.MessageService"/>
<component id="compToBeExported" scope="bundle"
class="com.xyz.MessageServiceImpl"/>
```

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.

```
<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
- `all-classes` : the service will be registered using the exported component's implementation type and super-types 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:

```
<service ref="componentToBeExported" auto-export="interfaces"/>
```

Given the interface hierarchy:

```
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 `org.osgi.service.context.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.

5.4.1.3 The depends-on attribute

As previously described, an exported service is always registered with the service property `org.osgi.service.context.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.

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).

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="register"> (3)
    <component class="SomeListenerClass"/>            (4)
  </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) Declare only a registration method for this listener.

(4) Listener component declared anonymously in-line.

The optional `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 one of the following formats:

```
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 registration/unregistration methods are only invoked when a service of a type compatible with the declared `ServiceType` is registered/unregistered.

5.4.2 Defining References to OSGi Services

This RFC 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. RFC 124 supports the management of this set of references as a List or Set.

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 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 `org.osgi.service.context.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.

Depends-on attribute

The optional `depends-on` attribute can be used to specify that the service reference should not be looked up in the service registry until the named dependent component has been instantiated.

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:

```
public class ComponentWithServiceReference {
    private ServiceReference serviceReference;
    private SomeService service;
    // getters/setters omitted
}
<reference id="service" interface="com.xyz.SomeService"/>
<component id="someComponent" class="ComponentWithServiceReference">
    <property name="serviceReference" ref="service"/>      (1)
    <property name="service" ref="service"/>              (2)
</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 the attributes `interface`, `filter`, `component-name`, and `availability` with the same semantics as 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:

```
<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. It is possible to specify a sorting order 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` element can be used to define an inline component. For example:

```
<ref-set id="myServices" interface="com.xyz.MyService"
  comparator-ref="someComparator"/>
```

```
<ref-list id="myOtherServices"
  interface="com.xyz.OtherService">
  <comparator>
    <component class="MyOtherServiceComparator"/>
  </comparator>
</ref-list>
```

To sort using a natural ordering instead of an explicit comparator, you can use the `natural-ordering` element inside of `comparator`. You need to specify the basis for the natural ordering: based on the service references, following the `ServiceReference` natural ordering defined in the OSGi Core Specification section 6.1.2.3; or based on the services themselves (in which case the services must be `Comparable`).

```
<ref-list id="myServices" interface="com.xyz.MyService">
  <comparator><natural-ordering basis="service"/></comparator>
</ref-list>
```

```
<ref-set id="myOtherServices" interface="com.xyz.OtherService">
  <comparator><natural-ordering basis="service-reference"/></comparator>
</ref-set>
```

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 property can also be declared as a `Map` type. In this case the property will be injected with a `Map` keyed by service id, with `ServiceReference` objects as the values.

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. 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 the reference becomes satisfied. The optional `timeout` attribute of the reference element enables a timeout value (in milliseconds) to be specified. If a timeout value is specified and no matching service becomes available within the timeout period, an unchecked `ServiceUnavailableException` is thrown.

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 passing a reference to M to A without any involvement from the container, then this dependency is not tracked.

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, 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:

```
<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:

```
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`.

5.4.3.3 Module-wide defaults for service references

Elements in the `osgi` namespace may optionally be enclosed in a top-level "osgi" element. This element supports the setting of `default-availability` and `default-timeout` attribute values that then serve as the defaults for the `availability` and `timeout` attributes of the `reference`, `ref-set`, and `ref-list` elements when no value is specified.

5.5 Module Context API

The `ModuleContext` interface provides access to the component objects within the module context and to metadata describing the components within the context. A component that implements the `ModuleContextAware` interface will be injected with an instance of `ModuleContext` during configuration.

5.6 Namespace Extension Mechanism

TODO; describe how additional namespaces are supported: see bug 694.

Outline solution: Namespace handlers are identified by the XML Schema URI of the schema that they support. A handler is simply a type that implements the `NamespaceHandler` interface. In addition a schema definition resource mapping can be used to specify a resource path to the associated xsd file within the bundle defining the handler

A manifest header is used to declare a namespace handler:

Module-Context-Namespace-Handler:

`http://www.mycompany.com/schema/myns;handler=com.xyz.Foo;schema=/com/xyz/foo.xsd`

The handler is available for use only for contexts created for the declaring bundle unless the “scope:=platform” directive is also specified.

The Module-Context-Namespace-Handler value is a comma-delimited list of handlers provided by the bundle. Each entry in the list names a schema URI, and the required handler and schema attributes referenced the namespace handler type and schema definition file within the bundle.

If multiple bundles declare a handler for the same URI, then the handler from the bundle with the lowest bundle id will be used.

Only see type-compatible namespace handlers:

- Bundle defining the namespace handler must be wired to the same exporter of the `org.osgi.module.context` package as the bundle defining the handler

TODO: define `NamespaceHandler` interface and all that goes with it (access to `ParserContext`, `BeanBuilder` interface etc.).

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:

```
<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. 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-service` 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 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.

```
<osgix:property-placeholder persistent-id="com.xyz.myapp">
  <default-properties>
    <property name="productCategory" value="E792"/>
    <property name="businessUnit" value="811"/>
  </default-properties>
</osgix:property-placeholder>
```

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 Managed Services

The `managed-service` element is used to define a component based on the configuration information stored under a given persistent id. It has two mandatory attributes, `class` and `persistent-id`. The `persistent-id` attribute is used to specify the persistent id to be looked up in the configuration administration service; `class` indicates the Java class of the component that will be instantiated.

A simple declaration of a managed service component looks as follows:

```
<osgix:managed-service id="myService" class="com.xyz.MessageService"
  persistent-id="com.xyz.messageservice"/>
```

The properties of the `managed-service` component are dependency injected by name (a component property "foo" will be injected with the value stored under key "foo" in the dictionary) based on the configuration found under the given persistent id. It is possible to declare regular component property elements within the `managed-service` declaration. If a property value is defined both in the configuration object stored in the Configuration Admin service, and in a nested property 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 autowire the component instance 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. Container-managed updates cannot be used in conjunction with constructor injection. Before proceeding to autowire based on the new property values, a lock is taken on the component instance. This lock is released once autowiring has completed. A class may therefore synchronize its service methods or otherwise lock on the component instance in order to have atomic update semantics.

5.7.3 Managed Service Factories

The `managed-service-factory` element is similar to the `managed-service` element, but instead defines a set of components, one instance for each configuration stored under the given factory pid. It has two mandatory attributes, `factory-pid` and `class`.

A simple `managed-service-factory` declaration looks as follows:

```
<osgix:managed-service-factory id="someId" factory-pid="org.xzy.services"
    class="MyServiceClass"/>
```

This declaration results in the creation of zero or more component instances, one instance for each configuration registered under the given factory pid. The components will have synthetic names generated by appending "-" followed by the persistent id of the configuration object as returned by Configuration Admin, to the value of the id attribute used in the declaration of the `managed-service-factory`. For example: `someId-config.admin.generated.pid`.

Over time new configuration objects may be added under the factory pid. A new component instance is automatically instantiated whenever a new configuration object is created. If a configuration object stored under the factory pid is deleted, then the corresponding component instance will be disposed. The optional `destroy-method` attribute of the `managed-service-factory` element may be used to specify a destroy callback to be invoked on the component instance. Such a method must have a signature:

```
public void anyMethodName();
```

It is also possible for the configuration of an existing component to be updated. The same `update-strategy` and `update-method` attributes are available as for the `managed-service` element and with the same semantics (though obviously only the component instance whose configuration has been updated in Configuration Admin will actually be updated). The same client-locking semantics also apply when using the `container-managed` update strategy.

5.7.4 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:

```
<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`.

5.7.5 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:

```
<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 `config-properties` element.

```
<service interface="org.osgi.service.cm.ManagedService" ref="MyManagedService">
```

```
  <service-properties>
```

```
    <osgi:config-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 update, specify the optional `update="true"` attribute value.

5.8 APIs

5.8.1 Package `org.osgi.module.context`

5.8.1.1 *ModuleContext*

```
package org.osgi.module.context;
```

```
import org.osgi.framework.BundleContext;
```

```
import org.osgi.module.context.reflect.ComponentMetadata;
```

```
import org.osgi.module.context.reflect.LocalComponentMetadata;
```

```
import org.osgi.module.context.reflect.ServiceExportComponentMetadata;
```

```
import org.osgi.module.context.reflect.ServiceReferenceComponentMetadata;
```

```
/**
```

```
 * 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.
```


Draft

```
* The Constants.BUNDLE_SYMBOLICNAME and Constants.BUNDLE_VERSION service
* properties can be used to determine which bundle the published ModuleContext
* service is associated with.
*
* @see ModuleContextAware
* @see org.osgi.framework.Constants
*
*/
public interface ModuleContext {

    /**
     * The names of all the named components within the module context.
     *
     * @return an array containing the names of all of the components within
     * the module.
     */
    String[] getComponentNames();

    /**
     * Get the component instance for a given named component.
     *
     * @param name the name of the component for which the instance is to be
retrieved
     *
     * @return the component instance, the type of the returned object is
dependent
the
     * on the component definition, and may be determined by introspecting
the
     * component metadata.
     *
     * @throws NoSuchNamedComponentException if the name specified is not the
name of a
     * component within the module.
     */
    Object getComponent(String name) throws NoSuchComponentException;

    /**
     * Get the component metadata for a given named component.
     *
     * @param name the name of the component for which the metadata is to be
retrieved.
     *
     * @return the component metadata for the component.
     *
     * @throws NoSuchNamedComponentException if the name specified is not the
name of a
     * component within the module.
     */
    ComponentMetadata getComponentMetadata(String name);

    /**
     * Get the service reference metadata for every OSGi service referenced
by
     * this module.
     */
}
```

Draft

```
service.
    * @return an array of metadata, with one entry for each referenced
be
    * If the module does not reference any services then an empty array will
    * returned.
    */
ServiceReferenceComponentMetadata[] getReferencedServicesMetadata();

/**
 * Get the service export metadata for every service exported by this
 * module.
 *
 * @return an array of metadata, with one entry for each service export.
 * If the module does not export any services then an empty array will be
 * returned.
 */
ServiceExportComponentMetadata[] getExportedServicesMetadata();

module.
    * Get the metadata for all components defined locally within this
will
    * @return an array of metadata, with one entry for each component.
    * If the module does not define any local components then an empty array
    * be returned.
    */
LocalComponentMetadata[] getLocalComponentsMetadata();

/**
 * Get the bundle context of the bundle this module context is
 * associated with.
 *
 * @return the module's bundle context
 */
BundleContext getBundleContext();
}
```

5.8.1.2 ModuleContextAware

```
package org.osgi.module.context;

/**
 * If a component implements this interface then the setModuleContext operation
 * will be invoked after the component instance has been instantiated and before
 * the init-method (if specified) has been invoked.
 *
 */
public interface ModuleContextAware {

    /**
```

Draft

```
    * Set the module context of the module in which the implementor is
    * executing.
    *
    * @param context the module context in which the implementor of
    * this interface is executing.
    */
void setModuleContext(ModuleContext context);
}
```

5.8.1.3 *ModuleContextListener*

```
package org.osgi.module.context;
public interface ModuleContextListener extends java.util.EventListener {
    void contextCreated(String bundleSymbolicName, org.osgi.framework.Version version);
    void contextCreationFailed(String bundleSymbolicName, org.osgi.framework.Version version,
        Throwable rootCause);
}
```

5.8.1.4 *ServiceUnavailableException*

```
package org.osgi.module.context;
public class ServiceUnavailableException extends RuntimeException {
    public ServiceUnavailableException(
        String message,
        Class serviceType,
        String filterExpression);
    public Class getServiceType();
    public String getFilter();
}
```

5.8.1.5 *NoSuchComponentException*

```
public class NoSuchComponentException extends RuntimeException {
    private final String componentName;
    public NoSuchComponentException(String componentName) {
        this.componentName = componentName;
    }
    public String getComponentName() {
        return this.componentName;
    }
    public String getMessage() {
        return "No component named '" +

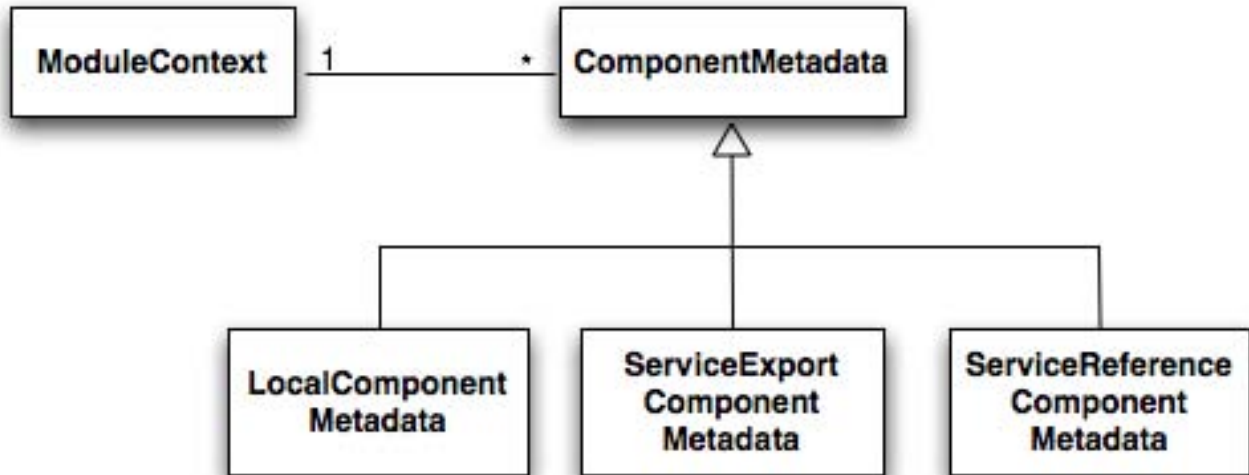
```

```

        (this.componentName == null ? "<null>" :
this.componentName) +
        "' could be found";
    }
}

```

5.8.2 Package org.osgi.module.context.reflect



5.8.2.1 ComponentMetadata

```

package org.osgi.module.context.reflect;

/**
 * Metadata for a component defined within a given module context.
 *
 * @see LocalComponentMetadata
 * @see ServiceReferenceComponentMetadata
 * @see ServiceExportComponentMetadata
 */
public interface ComponentMetadata {

    /**
     * The name of the component.
     *
     * @return component name. The component name may be null if this is an
    anonymously
     * defined inner component.
     */
    String getName();

    /**
     * Any aliases by which the component is also known.

```

Draft

```
    *
    * @return an array of aliases by which the component is known (does not
    * include the component name as returned by getName()). If the component
    * has no aliases then an empty array is returned.
    */
    String[] getAliases();

    /**
    * The names of any components listed in a "depends-on" attribute for
this
    * component.
    *
    * @return an array of component names for components that we have
explicitly
    * declared a dependency, or an empty array if none.
    */
    String[] getExplicitDependencies();
}
```

5.8.2.2 LocalComponentMetadata

```
package org.osgi.module.context.reflect;
```

```
/**
 * Metadata for a component defined locally with a module context.
 *
 */
public interface LocalComponentMetadata extends ComponentMetadata {

    static final String SCOPE_SINGLETON = "singleton";
    static final String SCOPE_PROTOTYPE = "prototype";
    static final String SCOPE_BUNDLE = "bundle";

    /**
    * The name of the class type specified for this component.
    *
    * @return the name of the component class.
    */
    String getClassName();

    /**
    * The name of the init method specified for this component, if any.
    *
    * @return the method name of the specified init method, or null if
    * no init method was specified.
    */
    String getInitMethodName();

    /**
    * The name of the destroy method specified for this component, if any.
    *
    * @return the method name of the specified destroy method, or null if no
    * destroy method was specified.
    */
}
```

Draft

```
    */
    String getDestroyMethodName();

    /**
     * The constructor injection metadata for this component.
     *
     * @return 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.
     */
    ConstructorInjectionMetadata getConstructorInjectionMetadata();

    /**
     * The property injection metadata for this component.
     *
     * @return an array containing one entry for each property to be
    injected. If
     * no property injection was specified for this component then an empty
    array
     * will be returned.
     */
    PropertyInjectionMetadata[] getPropertyInjectionMetadata();

    /**
     * The field injection metadata for this component.
     *
     * @return an array containing one entry for each field to be injected.
    If no
     * field injection was specified for this component then an empty array
    will be
     * returned.
     */
    FieldInjectionMetadata[] getFieldInjectionMetadata();

    /**
     * The method injection metadata for this component.
     *
     * @return an array containing one entry for each method to be invoked
    using method
     * injection after constructing the component instance. If no method
    injection
     * was specified for this component then an empty array will be returned.
     */
    MethodInjectionMetadata[] getMethodInjectionMetadata();

    /**
     * Is this an abstract component declaration.
     *
     * @return true, iff this component definition is marked as abstract and
    hence
     * has no associated component instance.
     */
    boolean isAbstract();
```

Draft

```
/**
 * Is this component to be lazily instantiated?
 *
 * @return true, iff this component definition specifies lazy
 * instantiation.
 */
boolean isLazy();

/**
 * The metadata for the parent definition of this component declaration,
if any.
 *
 * @return the component metadata for the parent component definition if
this component
 * was declared using component metadata inheritance.
 */
LocalComponentMetadata getParent();

/**
 * The metadata describing how to create the component instance by
invoking a
 * method (as opposed to a constructor) if factory methods are used.
 *
 * @return the method injection metadata for the specified factory
method, or null if no
 * factory method is used for this component.
 */
MethodInjectionMetadata getFactoryMethodMetadata();

/**
 * The component instance on which to invoke the factory method (if
specified).
 *
 * @return 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.
 */
ComponentMetadata getFactoryComponent();

/**
 * The specified scope for the component lifecycle.
 *
 * @return a String indicating the scope specified for the component.
 *
 * @see SCOPE_SINGLETON
 * @see SCOPE_PROTOTYPE
```

Draft

```
    * @see SCOPE_BUNDLE
    */
    String getScope();
}
```

5.8.2.3 ConstructorInjectionMetadata

```
package org.osgi.module.context.reflect;
```

```
/**
 * Metadata for a component defined locally with a module context.
 *
 */
public interface LocalComponentMetadata extends ComponentMetadata {

    static final String SCOPE_SINGLETON = "singleton";
    static final String SCOPE_PROTOTYPE = "prototype";
    static final String SCOPE_BUNDLE = "bundle";

    /**
     * The name of the class type specified for this component.
     *
     * @return the name of the component class.
     */
    String getClassName();

    /**
     * The name of the init method specified for this component, if any.
     *
     * @return the method name of the specified init method, or null if
     * no init method was specified.
     */
    String getInitMethodName();

    /**
     * The name of the destroy method specified for this component, if any.
     *
     * @return the method name of the specified destroy method, or null if no
     * destroy method was specified.
     */
    String getDestroyMethodName();

    /**
     * The constructor injection metadata for this component.
     *
     * @return 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.
     */
    ConstructorInjectionMetadata getConstructorInjectionMetadata();

}
```


Draft

```

    * The property injection metadata for this component.
    *
    * @return an array containing one entry for each property to be
injected. If
    * no property injection was specified for this component then an empty
array
    * will be returned.
    *
    */
PropertyInjectionMetadata[] getPropertyInjectionMetadata();

/**
 * The field injection metadata for this component.
 *
 * @return an array containing one entry for each field to be injected.
If no
 * field injection was specified for this component then an empty array
will be
 * returned.
 *
 */
FieldInjectionMetadata[] getFieldInjectionMetadata();

/**
 * The method injection metadata for this component.
 *
 * @return an array containing one entry for each method to be invoked
using method
 * injection after constructing the component instance. If no method
injection
 * was specified for this component then an empty array will be returned.
 */
MethodInjectionMetadata[] getMethodInjectionMetadata();

/**
 * Is this an abstract component declaration.
 *
 * @return true, iff this component definition is marked as abstract and
hence
 * has no associated component instance.
 */
boolean isAbstract();

/**
 * Is this component to be lazily instantiated?
 *
 * @return true, iff this component definition specifies lazy
 * instantiation.
 */
boolean isLazy();

/**
 * The metadata for the parent definition of this component declaration,
if any.
 *
```

Draft

```
    * @return the component metadata for the parent component definition if
this component
    * was declared using component metadata inheritance.
    */
    LocalComponentMetadata getParent();

    /**
    * The metadata describing how to create the component instance by
invoking a
    * method (as opposed to a constructor) if factory methods are used.
    *
    * @return the method injection metadata for the specified factory
method, or null if no
    * factory method is used for this component.
    */
    MethodInjectionMetadata getFactoryMethodMetadata();

    /**
    * The component instance on which to invoke the factory method (if
specified).
    *
    * @return 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.
    */
    ComponentMetadata getFactoryComponent();

    /**
    * The specified scope for the component lifecycle.
    *
    * @return a String indicating the scope specified for the component.
    *
    * @see SCOPE_SINGLETON
    * @see SCOPE_PROTOTYPE
    * @see SCOPE_BUNDLE
    */
    String getScope();
}
```

5.8.2.4 PropertyInjectionMetadata

```
package org.osgi.module.context.reflect;
```

```
/**
 * Metadata describing a property to be injected. Properties are defined
```

Draft

```
* following JavaBeans conventions.
*/
public interface PropertyInjectionMetadata {

    /**
     * The name of the property to be injected, following JavaBeans
conventions.
     *
     * @return the property name.
     */
    String getName();

    /**
     * The value to inject the property with.
     *
     * @return the property value.
     */
    Value getValue();
}
```

5.8.2.5 *FieldInjectionMetadata*

```
package org.osgi.module.context.reflect;
```

```
/**
 * Metadata describing a field of a component that is to be injected.
 */
public interface FieldInjectionMetadata {

    /**
     * The name of the field to be injected.
     *
     * @return the field name
     */
    String getName();

    /**
     * The value to inject the field with.
     *
     * @return the field value
     */
    Value getValue();
}
```

5.8.2.6 *MethodInjectionMetadata*

```
package org.osgi.module.context.reflect;
```

```
/**
```

Draft

```
* Metadata describing a method to be invoked as part of component configuration.
*
*/
public interface MethodInjectionMetadata {

    /**
     * The name of the method to be invoked.
     *
     * @return the method name, overloaded methods are disambiguated by
     * parameter specifications.
     */
    String getName();

    /**
     * The parameter specifications that determine which method to invoke
     * (in the case of overloading) and what arguments to pass to it.
     *
     * @return an array of parameter specifications, or an empty array if the
     * method takes no arguments.
     */
    ParameterSpecification[] getParameterSpecifications();
}

```

5.8.2.7 ParameterSpecification

```
package org.osgi.module.context.reflect;

/**
 * Metadata describing a parameter of a method or constructor and the
 * value that is to be passed during injection.
 *
 * @see NamedParameterSpecification
 * @see TypedParameterSpecification
 * @see IndexedParameterSpecification
 */
public interface ParameterSpecification {

    /**
     * The value to inject into the parameter.
     *
     * @return the parameter value
     */
    Value getValue();
}

```

5.8.2.8 TypedParameterSpecification

```
package org.osgi.module.context.reflect;

/**
 * Parameter specification for injection of a parameter by type.

```

Draft

```
*/
public interface TypedParameterSpecification extends ParameterSpecification {

    /**
     * The name of the type that the parameter type must be assignable from.
     *
     * @return the parameter type name
     */
    String getTypeName();

}
```

5.8.2.9 IndexedParameterSpecification

```
package org.osgi.module.context.reflect;
```

```
/**
 * Parameter specification for injection of a parameter identified by its position
 * in the
 * argument list.
 *
 */
public interface IndexedParameterSpecification extends ParameterSpecification {

    /**
     * The index into the argument list of the parameter to be injected.
     *
     * @return the parameter index, indices start at 0.
     */
    int getIndex();

}
```

5.8.2.10 NamedParameterSpecification

```
package org.osgi.module.context.reflect;
```

```
/**
 * Parameter specification for injection of a parameter by name.
 *
 */
public interface NamedParameterSpecification extends ParameterSpecification {

    /**
     * The name of the parameter to be injected.
     *
     * @return the parameter name
     */
    String getName();

}
```

5.8.2.11 Value

```
package org.osgi.module.context.reflect;

/**
 * A value to inject into a field, property, method argument or constructor
 * argument.
 */
public interface Value {
    // deliberately left empty
}
```

5.8.2.12 ComponentValue

```
package org.osgi.module.context.reflect;

/**
 * A value represented by an anonymous local component definition.
 */
public interface ComponentValue extends Value {

    LocalComponentMetadata getComponentMetadata();

}
```

5.8.2.13 ReferenceValue

```
package org.osgi.module.context.reflect;

/**
 * A value which refers to another component in the module context by name.
 */
public interface ReferenceValue extends Value {

    /**
     * The name of the referenced component.
     */
    String getComponentName();

}
```

5.8.2.14 ReferenceNameValue

```
package org.osgi.module.context.reflect;

/**
 * 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.
 *
 */
```

Draft

```
public interface ReferenceNameValue extends Value {  
    String getReferenceName();  
}
```

5.8.2.15 MapValue

```
package org.osgi.module.context.reflect;  
  
import java.util.Map;  
  
/**  
 * A map-based value. Map keys are instances of Value, as are the Map entry  
 * values themselves.  
 */  
public interface MapValue extends Value, Map/*<Value,Value>*/ {  
}
```

5.8.2.16 SetValue

```
package org.osgi.module.context.reflect;  
  
import java.util.Set;  
  
/**  
 * A set-based value. Members of the set are instances of Value.  
 */  
public interface SetValue extends Value, Set/*<Value>*/ {  
}
```

5.8.2.17 ListValue

```
package org.osgi.module.context.reflect;  
  
import java.util.List;  
  
/**  
 * A list-based value. Members of the List are instances of Value.  
 */  
public interface ListValue extends Value, List/*<Value>*/ {  
}
```

5.8.2.18 PropertiesValue

```
package org.osgi.module.context.reflect;

import java.util.Properties;

/**
 * A properties-based value.
 */
public interface PropertiesValue extends Value, Properties {

}
```

5.8.2.19 NullValue

```
package org.osgi.module.context.reflect;

/**
 * A null value.
 */
public interface NullValue extends Value{

}
```

5.8.2.20 TypedStringValue

```
package org.osgi.module.context.reflect;

/**
 * A simple string value that will be type-converted if necessary before
 * injecting into a target.
 */
public interface TypedStringValue extends Value {

    /**
     * The string value (unconverted) of this value).
     */
    String getStringValue();

    /**
     * The name of the type to which this value should be coerced. May be
     null.
     */
    String getTypeName();

}
```


5.8.2.21 ServiceExportComponentMetadata

```
package org.osgi.module.context.reflect;
```

```
import java.util.Properties;
```

```
/**
```

```
 * Metadata representing a service to be exported by a module context.
```

```
 *
```

```
 */
```

```
public interface ServiceExportComponentMetadata extends ComponentMetadata {
```

```
    /**
```

```
     * Do not auto-detect types for advertised service interfaces
```

```
     */
```

```
    public static final int EXPORT_MODE_DISABLED = 1;
```

```
    /**
```

```
     * Advertise all Java interfaces implemented by the exported component as
```

```
     * service interfaces.
```

```
     */
```

```
    public static final int EXPORT_MODE_INTERFACES = 2;
```

```
    /**
```

```
     * Advertise all Java classes in the hierarchy of the exported  
component's type
```

```
     * as service interfaces.
```

```
     */
```

```
    public static final int EXPORT_MODE_CLASS_HIERARCHY = 3;
```

```
    /**
```

```
     * Advertise all Java classes and interfaces in the exported component's  
type as
```

```
     * service interfaces.
```

```
     */
```

```
    public static final int EXPORT_MODE_ALL = 4;
```

```
    /**
```

```
     * The component that is to be exported as a service. Value must refer to  
a component and
```

```
     * therefore be either a ComponentValue, ReferenceValue, or
```

```
ReferenceNameValue.
```

```
     *
```

```
     * @return the component to be exported as a service.
```

```
     */
```

```
    Value getExportedComponent();
```

```
    /**
```

```
     * The type names of the set of interface types that the service should  
be advertised
```

```
     * as supporting.
```

```
     *
```

Draft

```
    * @return an array of type names, or an empty array if using auto-export
    */
    String[] getInterfaceNames();

    /**
     * Return the auto-export mode specified.
     *
     * @return One of EXPORT_MODE_DISABLED, EXPORT_MODE_INTERFACES,
     EXPORT_MODE_CLASS_HIERARCHY, EXPORT_MODE_ALL
     */
    int getAutoExportMode();

    /**
     * The user declared properties to be advertised with the service.
     *
     * @return Properties object containing the set of user declared service
properties (may be
     * empty if no properties were specified).
     */
    Properties getServiceProperties();

    /**
     * The ranking value to use when advertising the service
     *
     * @return service ranking
     */
    int getRanking();

    /**
     * The listeners that have registered to be notified when the exported
service
     * is registered and unregistered with the framework.
     *
     * @return an array of registration listeners, or an empty array if no
listeners
     * have been specified.
     */
    RegistrationListenerMetadata[] getRegistrationListeners();
}

```

5.8.2.2 *RegistrationListenerMetadata*

```
package org.osgi.module.context.reflect;

/**
 * Metadata for a listener interested in service registration and unregistration
 * events for an exported service.
 */
public interface RegistrationListenerMetadata {

    /**

```

Draft

```
    * 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.
    *
    * @return the listener component reference.
    */
Value getListenerComponent();

/**
 * The name of the method to invoke on the listener component when
 * the exported service is registered with the service registry.
 *
 * @return the registration callback method name.
 */
String getRegistrationMethodName();

/**
 * The name of the method to invoke on the listener component when
 * the exported service is unregistered from the service registry.
 *
 * @return the unregistration callback method name.
 */
String getUnregistrationMethodName();
}
```

5.8.2.23 ServiceReferenceComponentMetadata

```
package org.osgi.module.context.reflect;
```

```
/**
 * Metadata describing a reference to a service that is to be imported into the
 * module
 * context from the OSGi service registry.
 */
public interface ServiceReferenceComponentMetadata extends ComponentMetadata {

    /**
     * A matching service is required at all times.
     */
    public static final int AVAILABILITY_MANDATORY = 1;

    /**
     * A matching service is not required to be present.
     */
    public static final int AVAILABILITY_OPTIONAL = 2;

    /**
     * Whether or not a matching service is required at all times.
     *
     * @return one of MANDATORY_AVAILABILITY or OPTIONAL_AVAILABILITY
     */
}
```

Draft

```
    */
    int getServiceAvailabilitySpecification();

    /**
     * The interface types that the matching service must support
     *
     * @return an array of type names
     */
    String[] getInterfaceNames();

    /**
     * The filter expression that a matching service must pass
     *
     * @return filter expression
     */
    String getFilterString();

    /**
     * The set of listeners registered to receive bind and unbind events for
     * backing services.
     *
     * @return an array of registered binding listeners, or an empty array
     * if no listeners are registered.
     */
    BindingListenerMetadata[] getBindingListeners();
}

```

5.8.2.24 *UnaryServiceReferenceComponentMetadata*

```
package org.osgi.module.context.reflect;
```

```
/**
 *
 * Service reference that will bind to a single matching service
 * in the service registry.
 *
 */
public interface UnaryServiceReferenceComponentMetadata extends
    ServiceReferenceComponentMetadata {

    /**
     * Timeout for service invocations when a matching backing service
     * is unavailable.
     *
     * @return service invocation timeout in milliseconds
     */
    long getTimeout();
}

```

Draft

5.8.2.25 *CollectionBasedServiceReferenceComponentMetadata*

```
package org.osgi.module.context.reflect;
```

```
/**
 * Service reference that binds to a collection of matching services from
 * the OSGi service registry.
 */
public interface CollectionBasedServiceReferenceComponentMetadata extends
    ServiceReferenceComponentMetadata {

    /**
     * Create natural ordering based on comparison on service objects.
     */
    public static final int ORDER_BASIS_SERVICES = 1;

    /**
     * Create natural ordering based on comparison of service reference
    objects.
     */
    public static final int ORDER_BASIS_SERVICE_REFERENCES = 2;

    /**
     * Track matching services in a managed Set
     */
    public static final int COLLECTION_TYPE_SET = 1;

    /**
     * Track matching services in a managed List
     */
    public static final int COLLECTION_TYPE_LIST = 2;

    /**
     * The type of collection to be created.
     *
     * @return one of COLLECTION_TYPE_SET or COLLECTION_TYPE_LIST
     */
    int getCollectionType();

    /**
     * The comparator specified for ordering the collection, or null if no
     * comparator was specified.
     *
     * @return 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.
     */
    Value getComparator();

    /**
     * Should the collection be ordered based on natural ordering?
     */
}
```

Draft

```
    * @return true, iff natural-ordering based sorting was specified.
    */
    boolean isNaturalOrderingBasedComparison();

    /**
     * The basis on which to perform natural ordering, if specified.
     *
     * @return one of ORDER_BASIS_SERVICES and ORDER_BASIS_SERVICE_REFERENCES
     */
    int getNaturalOrderingComparisonBasis();
}
```

5.8.2.26 *BindingListenerMetadata*

```
package org.osgi.module.context.reflect;

/**
 * Metadata for a listener interested in service bind and unbind events for a
 * service
 * reference.
 */
public interface BindingListenerMetadata {

    /**
     * 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.
     *
     * @return the listener component reference.
     */
    Value getListenerComponent();

    /**
     * The name of the method to invoke on the listener component when
     * a matching service is bound to the reference
     *
     * @return the bind callback method name.
     */
    String getBindMethodName();

    /**
     * The name of the method to invoke on the listener component when
     * a service is unbound from the reference.
     *
     * @return the unbind callback method name.
     */
    String getUnbindMethodName();
}
```

5.9 'osgi' Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.osgi.org/schema/context"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.osgi.org/schema/context"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  version="1.0-rc2">

  <!-- Schema elements for core component declarations -->

  <xsd:complexType name="identifiedType" 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:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="component"/>
          <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>
      <xsd:attribute name="default-init-method" type="xsd:string">
      </xsd:attribute>
      <xsd:attribute name="default-destroy-method" type="xsd:string">
      </xsd:attribute>
      <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:complexType>
  </xsd:element>

  <xsd:group name="componentElements">
    <xsd:sequence>
      <xsd:element ref="description" minOccurs="0"/>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="constructor-arg"/>
        <xsd:element ref="property"/>
        <xsd:any namespace="##other" processContents="strict" minOccurs="0"
          maxOccurs="unbounded"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:group>

  <xsd:attributeGroup name="componentAttributes">
    <xsd:attribute name="name" type="xsd:string">
    </xsd:attribute>
    <xsd:attribute name="class" type="xsd:string">
    </xsd:attribute>
    <xsd:attribute name="parent" type="xsd:string">
    </xsd:attribute>
    <xsd:attribute name="scope" type="xsd:string">
    </xsd:attribute>
    <xsd:attribute name="abstract" type="xsd:boolean">
  </xsd:attributeGroup>
```

Draft

```
</xsd:attribute>
<xsd:attribute name="lazy-init" default="default" type="defaultable-boolean">
</xsd:attribute>
<xsd:attribute name="depends-on" type="xsd:string">
</xsd:attribute>
<xsd:attribute name="init-method" type="xsd:string">
</xsd:attribute>
<xsd:attribute name="destroy-method" type="xsd:string">
</xsd:attribute>
<xsd:attribute name="factory-method" type="xsd:string">
</xsd:attribute>
<xsd:attribute name="factory-component" type="xsd:string">
</xsd:attribute>
<xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:attributeGroup>

<xsd:element name="component">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="identifiedType">
        <xsd:group ref="componentElements" />
        <xsd:attributeGroup ref="componentAttributes" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name="constructor-arg">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="description" minOccurs="0" />
      <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="props" />
        <xsd:any namespace="##other" processContents="strict" />
      </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="index" type="xsd:string">
    </xsd:attribute>
    <xsd:attribute name="type" type="xsd:string">
    </xsd:attribute>
    <xsd:attribute name="ref" type="xsd:string">
    </xsd:attribute>
    <xsd:attribute name="value" type="xsd:string">
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>

<xsd:element name="property" type="propertyType">
</xsd:element>

<xsd:element name="arg-type">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:attribute name="match" type="xsd:string">
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ref">
```


Draft

```
<xsd:complexType>
  <xsd:complexContent>
    <xsd:restriction base="xsd:anyType">
      <xsd:attribute name="component" type="xsd:string">
      </xsd:attribute>
      <xsd:attribute name="local" type="xsd:IDREF">
      </xsd:attribute>
      <xsd:attribute name="parent" type="xsd:string">
      </xsd:attribute>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
</xsd:element>

<xsd:element name="idref">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:attribute name="component" type="xsd:string">
        </xsd:attribute>
        <xsd:attribute name="local" type="xsd:IDREF">
        </xsd:attribute>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name="value">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded"/>
    <xsd:attribute name="type" type="xsd:string">
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>

<xsd:element name="null">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded"/>
  </xsd:complexType>
</xsd:element>

<!-- Collection Elements -->
<xsd:group name="collectionElements">
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <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="props"/>
      <xsd:any namespace="##other" processContents="strict"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:group>

<xsd:element name="list" type="listOrSetType">
</xsd:element>

<xsd:element name="set" type="listOrSetType">
</xsd:element>

<xsd:element name="map" type="mapType">
```

Draft

```
</xsd:element>

<xsd:element name="entry" type="entryType">
</xsd:element>

<xsd:element name="props" type="propsType">
</xsd:element>

<xsd:element name="key">
  <xsd:complexType>
    <xsd:group ref="collectionElements" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="prop">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded" />
    <xsd:attribute name="key" type="xsd:string" use="required">
</xsd:attribute>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="propertyType">
  <xsd:sequence>
    <xsd:element ref="description" minOccurs="0" />
    <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="props" />
      <xsd:any namespace="##other" processContents="strict" />
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required">
</xsd:attribute>
  <xsd:attribute name="ref" type="xsd:string">
</xsd:attribute>
  <xsd:attribute name="value" type="xsd:string">
</xsd:attribute>
</xsd:complexType>

<!-- Collection Types -->

<!-- base collection type -->
<xsd:complexType name="baseCollectionType">
</xsd:complexType>

<!-- base type for collections that have (possibly) typed nested values -->
<xsd:complexType name="typedCollectionType">
  <xsd:complexContent>
    <xsd:extension base="baseCollectionType">
      <xsd:attribute name="value-type" type="xsd:string">
</xsd:attribute>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- 'map' element type -->
<xsd:complexType name="mapType">
  <xsd:complexContent>
    <xsd:extension base="typedCollectionType">
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">

```

Draft

```
        <xsd:element ref="entry" />
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>

<!-- 'entry' element type -->
<xsd:complexType name="entryType">
  <xsd:sequence>
    <xsd:element ref="key" minOccurs="0" />
    <xsd:group ref="collectionElements" />
  </xsd:sequence>
  <xsd:attribute name="key" type="xsd:string">
  </xsd:attribute>
  <xsd:attribute name="key-ref" type="xsd:string">
  </xsd:attribute>
  <xsd:attribute name="value" type="xsd:string">
  </xsd:attribute>
  <xsd:attribute name="value-ref" type="xsd:string">
  </xsd:attribute>
</xsd:complexType>

<!-- 'list' and 'set' collection type -->
<xsd:complexType name="listOrSetType">
  <xsd:complexContent>
    <xsd:extension base="typedCollectionType">
      <xsd:group ref="collectionElements" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- 'props' collection type -->
<xsd:complexType name="propsType">
  <xsd:complexContent>
    <xsd:extension base="baseCollectionType">
      <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="defaultable-boolean">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="default" />
    <xsd:enumeration value="true" />
    <xsd:enumeration value="false" />
  </xsd:restriction>
</xsd:simpleType>

<!-- Elements from Spring Dynamic Modules project -->

<xsd:attributeGroup name="defaults">
  <xsd:attribute name="default-timeout" type="xsd:long" default="30000"/>
  <xsd:attribute name="default-availability" type="Tavailability" default="mandatory"/>
</xsd:attributeGroup>

<xsd:simpleType name="TdefaultCardinalityOptions">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="1..X" />
    <xsd:enumeration value="0..X" />
  </xsd:restriction>
</xsd:simpleType>
```

Draft

```
</xsd:simpleType>

<!-- reference -->
<xsd:element name="reference" type="TsingleReference"/>

<xsd:complexType name="Treference">
  <xsd:complexContent>
    <xsd:extension base="identifiedType">
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="interfaces" type="listOrSetType" 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="depends-on" type="xsd:string" use="optional"/>
      <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:sequence minOccurs="0" maxOccurs="1">
    <!-- nested component declaration -->
    <xsd:any namespace="##other" minOccurs="1" maxOccurs="1" processContents="skip"/>
  </xsd:sequence>

  <!-- shortcut for bean references -->
  <xsd:attribute name="ref" type="xsd:string" use="optional"/>
  <xsd:attribute name="bind-method" type="xsd:token" use="optional"/>
  <xsd:attribute name="unbind-method" type="xsd:token" use="optional"/>
</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="required"/>
    <xsd:enumeration value="optional"/>
  </xsd:restriction>
</xsd:simpleType>

<!-- reference collections (set, list) -->
<xsd:element name="ref-list" type="TreferenceCollection"/>

<xsd:element name="ref-set" type="TreferenceCollection"/>

<xsd:complexType name="TreferenceCollection">
  <xsd:complexContent>
    <xsd:extension base="Treference">
      <xsd:sequence minOccurs="0" maxOccurs="1">
        <xsd:element name="comparator" type="Tcomparator"/>
      </xsd:sequence>
      <xsd:attribute name="comparator-ref" type="xsd:string" use="optional"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Draft

```
<xsd:complexType name="Tcomparator">
  <xsd:choice>
    <xsd:element name="natural" type="TnaturalOrdering"/>
    <xsd:sequence minOccurs="1" maxOccurs="1">
      <!-- nested bean declaration -->
      <xsd:any namespace="##other" minOccurs="1" maxOccurs="1" processContents="skip"/>
    </xsd:sequence>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="TnaturalOrdering">
  <xsd:attribute name="basis" type="TorderingBasis" use="required"/>
</xsd:complexType>

<xsd:simpleType name="TorderingBasis">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="service"/>
    <xsd:enumeration value="service-reference"/>
  </xsd:restriction>
</xsd:simpleType>

<!-- service -->
<xsd:element name="service" type="Tservice"/>

<xsd:complexType name="Tservice">
  <xsd:complexContent>
    <xsd:extension base="identifiedType">
      <xsd:sequence minOccurs="0" maxOccurs="1">
        <xsd:element name="interfaces" type="listOrSetType" minOccurs="0"/>
        <xsd:element name="service-properties" minOccurs="0" type="mapType"/>
        <xsd:element name="registration-listener" type="TserviceRegistrationListener"
          minOccurs="0" maxOccurs="unbounded"/>
        <!-- nested bean declaration -->
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="1" processContents="skip"/>
      </xsd:sequence>
      <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:complexType name="TserviceRegistrationListener">
  <xsd:sequence minOccurs="0" maxOccurs="1">
    <!-- nested bean declaration -->
    <xsd:any namespace="##other" minOccurs="1" maxOccurs="1" processContents="skip"/>
  </xsd:sequence>

  <!-- shortcut for bean references -->
  <xsd:attribute name="ref" type="xsd:string" use="optional"/>
  <xsd:attribute name="registration-method" type="xsd:token" use="optional"/>
  <xsd:attribute name="unregistration-method" type="xsd:token" use="optional"/>
</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>

</xsd:schema>
```

5.10 'osgix' Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.osgi.org/schema/osgi-compendium"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:osgi="http://www.osgi.org/schema/context"
  targetNamespace="http://www.osgi.org/schema/osgi-compendium"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  version="1.0-rc2">

  <xsd:import namespace="http://www.osgi.org/schema/context" />

  <!-- property placeholder -->

  <xsd:element name="property-placeholder" type="TpropertyPlaceholder" />

  <xsd:complexType name="TpropertyPlaceholder">
    <xsd:complexContent>
      <xsd:extension base="osgi:identifiedType">
        <xsd:sequence minOccurs="0" maxOccurs="1">
          <!-- nested properties declaration -->
          <xsd:element name="default-properties" type="osgi:propsType" minOccurs="0" maxOccurs="1" />
        </xsd:sequence>
        <xsd:attribute name="persistent-id" type="xsd:string" use="required" />
        <xsd:attribute name="placeholder-prefix" type="xsd:string" use="optional" default="{"/>
        <xsd:attribute name="placeholder-suffix" type="xsd:string" use="optional" default="}"/>
        <xsd:attribute name="defaults-ref" type="xsd:string" use="optional" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- managed-service -->

  <xsd:element name="managed-service" type="TmanagedService" />

  <xsd:complexType name="TmanagedService">
    <xsd:complexContent>
      <xsd:extension base="osgi:identifiedType">
        <xsd:group ref="osgi:beanElements" />
        <xsd:attributeGroup ref="osgi:beanAttributes" />
        <xsd:attribute name="persistent-id" type="xsd:string" use="required" />
        <xsd:attribute name="updateStrategy" type="TupdateStrategyType" use="optional" />
        <xsd:attribute name="update-method" type="xsd:string" use="optional" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- managed-service-factory -->

  <xsd:element name="managed-service-factory" type="TmanagedServiceFactory" />

  <xsd:complexType name="TmanagedServiceFactory">
    <xsd:complexContent>
      <xsd:extension base="osgi:identifiedType">
        <xsd:group ref="osgi:beanElements" />
        <xsd:attributeGroup ref="osgi:beanAttributes" />
        <xsd:attribute name="factory-pid" type="xsd:string" use="required" />
        <xsd:attribute name="updateStrategy" type="TupdateStrategyType" use="optional" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:simpleType name="TupdateStrategyType">
    <xsd:restriction base="xsd:string">
```

```
<xsd:enumeration value="none" />
<xsd:enumeration value="bean-managed" />
<xsd:enumeration value="container-managed" />
</xsd:restriction>
</xsd:simpleType>

<!-- config-properties -->

<xsd:element name="config-properties" type="TconfigProperties" />

<xsd:complexType name="TconfigProperties">
  <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.

7 Security Considerations

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

8 Document Support

8.1 References

- [1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.
- [2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

8.2 Author's Address

Name	Adrian Colyer
Company	SpringSource
Address	Kenneth Dibben House Enterprise Road Chilworth Southampton SO16 7NS ENGLAND
Voice	+44 2380 111500
e-mail	adrian.colyer@springsource.com

8.3 Acronyms and Abbreviations

8.4 End of Document



The OSGi Alliance and its members specify, create, advance, and promote wide industry adoption of an open delivery and management platform for application services in home, commercial buildings, automotive and industrial environments. The OSGi Alliance serves as the focal point for a collaborative ecosystem of service providers, developers, manufacturers, and consumers. The OSGi specifications define a standardized, component oriented, computing environment for networked services. OSGi technology is currently being delivered in products and services shipping from several Fortune 100 companies. The OSGi Alliance's horizontal software integration platform is ideal for both vertical and cross-industry business models within home, vehicle, mobile and industrial environments. As an independent non-profit corporation, the OSGi Alliance also provides for the fair and uniform creation and distribution of relevant intellectual property – including specifications, reference implementations, and test suites – to all its members.

HOW TO REACH US:

OSGi Alliance
Bishop Ranch 6
2400 Camino Ramon, Suite 375
San Ramon, CA 94583 USA

Phone: +1.925.275.6625
E-mail: marketinginfo@osgi.org
Web: <http://www.osgi.org>

OSGi is a trademark of the OSGi Alliance in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

All other marks are trademarks of their respective companies.