RFC-0112 Bundle Repository

Confidential, Draft

38 Pages

Abstract

This RFC describes a bundle repository for the OSGi Alliance. This repository consists of a web site (bundles.osgi.org) that hosts an XML resource that describes a federated repository under the control of the OSGi Alliance. This repository can be browsed on the web site. Additionally, the repository can be used directly from any OSGi Framework to deploy bundles from the repository (if the bundles do not require licensing). This RFC defines the format of the XML and the OSGi Framework service to access and use the repository.

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 5

1 Introduction **5**
1.1 Acknowledgements 5
1.2 Introduction 5

2 Application Domain **5**

3 Problem Description **7**

4 Requirements **7**
4.1 Functional 7
4.2 Discovery 7
4.3 Dependency Resolution 7
4.4 Security 8
4.5 Non Functional 8

5 Technical Solution **8**
5.1 Entities 8
5.2 Overview 10
5.2.1 Resource 11
5.2.2 Capabilties 12
5.2.3 Requirements 12
5.2.4 Extends 13
5.3 Repository Admin 13
5.3.1 Discovery 13
5.3.2 Resolving 13
5.3.3 Admin 14
5.4 Resolving 14
5.5 XML Schema 14
5.5.1 Namespace 14
5.5.2 The XML Structure 14
5.5.3 Repository 14
5.5.4 Referral 15
5.5.5 Resource 15

All Page Within This Box

5.5.6 Category....15
5.5.7 Require....16
5.5.8 Extend....16
5.5.9 Capability....17
5.6 Filter Extensions....17
5.6.1 Greater and Less Operators....17
5.6.2 Set Arithmetic....17
5.6.3 Version Ranges....18
5.7 Sample XML File....18
5.8 Querying a Web Service Based Repository....19
5.9 Bundle Manifest Header Mapping....19
5.9.1 Bundle....19
5.9.2 Import and Export Package: 'package'....20
5.9.3 Require-Bundle....20
5.9.4 Fragment-Host....21
5.9.5 Import- and Export-Service....21
5.9.6 Declarative Services....22
5.9.7 Bundle-ExecutionEnvironment....22

6 Open Issues....22
6.1 Uses Constraint....22
6.2 Query protocol....22
6.3 Licensing....22

7 Java Documentation....23
7.1 Package org.osgi.servicex.obr....23
7.2 org.osgi.servicex.obr Interface Resource....23
7.2.1 LICENSE_URI....25
7.2.2 DESCRIPTION....25
7.2.3 DOCUMENTATION_URI....25
7.2.4 COPYRIGHT....25
7.2.5 SOURCE_URI....25
7.2.6 SYMBOLIC_NAME....26
7.2.7 PRESENTATION_NAME....26
7.2.8 ID....26
7.2.9 VERSION....26
7.2.10 URI....26
7.2.11 SIZE....26
7.2.12 KEYS....26
7.2.13 getProperties....26
7.2.14 getSymbolicName....27
7.2.15 getPresentationName....27
7.2.16 getVersion....27
7.2.17 getId....27
7.2.18 getURI....27
7.2.19 getRequirements....27
7.2.20 getRequests....27
7.2.21 getExtends....27
7.2.22 getCapabilities....27
7.2.23 getCategories....27
7.2.24 getRepository....27
7.3 org.osgi.servicex.obr Interface Resolver....28

All Page Within This Box

7.3.1 add 28
7.3.2 getUnsatisfiedRequirements 28
7.3.3 getOptionalResources....29
7.3.4 getReason 29
7.3.5 getResources 29
7.3.6 getRequiredResources 29
7.3.7 getAddedResources 29
7.3.8 resolve....29
7.3.9 deploy....29
7.4 org.osgi.servicex.obr Interface Requirement....29
7.4.1 getName....30
7.4.2 getFilter 30
7.4.3 isMultiple 30
7.4.4 isOptional 30
7.4.5 getComment....30
7.4.6 isSatisfied....31
7.5 org.osgi.servicex.obr Class RepositoryPermission....31
7.5.1 RepositoryPermission 32
7.6 org.osgi.servicex.obr Interface RepositoryAdmin 32
7.6.1 discoverResources 33
7.6.2 resolver....33
7.6.3 addRepository 33
7.6.4 removeRepository....34
7.6.5 listRepositories 34
7.6.6 getResource 34
7.7 org.osgi.servicex.obr Interface Repository 34
7.7.1 getURL 35
7.7.2 getResources 35
7.7.3 getName....35
7.7.4 getLastModified 35
7.8 org.osgi.servicex.obr Interface CapabilityProvider....35
7.8.1 getCapabilities....36
7.9 org.osgi.servicex.obr Interface Capability....36
7.9.1 getName....37
7.9.2 getProperties 37

8 Security Considerations....37
8.1 Repository Permission....37

9 Document Support 38
9.1 References 38
9.2 Author's Address....38
9.3 Acronyms and Abbreviations....38
9.4 End of Document 38

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].

All Page Within This Box

`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	DEC 22 2005	Peter Kriens, aQute, Initial draft

1 Introduction

1.1 Acknowledgements

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

1.2 Introduction

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

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

2 Application Domain

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

All Page Within This Box

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

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

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

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

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

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

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

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

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

3 Problem Description

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

4 Requirements

4.1 Functional

- Provide browsing access to a bundle repository via a web server
- Provide access to a bundle repository so that bundles can be directly installed after discovery
- Handle dependency resolution so that bundles can be deployed without generating errors
- Allow repositories to be linked, creating a federated repository
- Provide programmatic (service) access to the repository

4.2 Discovery

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

4.3 Dependency Resolution

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

4.4 Security

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

4.5 Non Functional

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

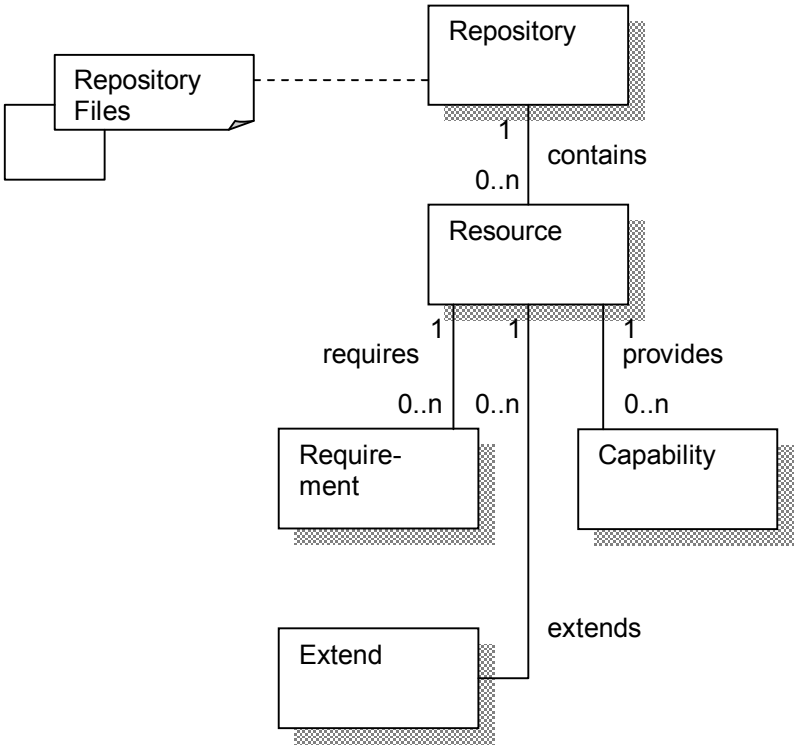
5 Technical Solution

5.1 Entities

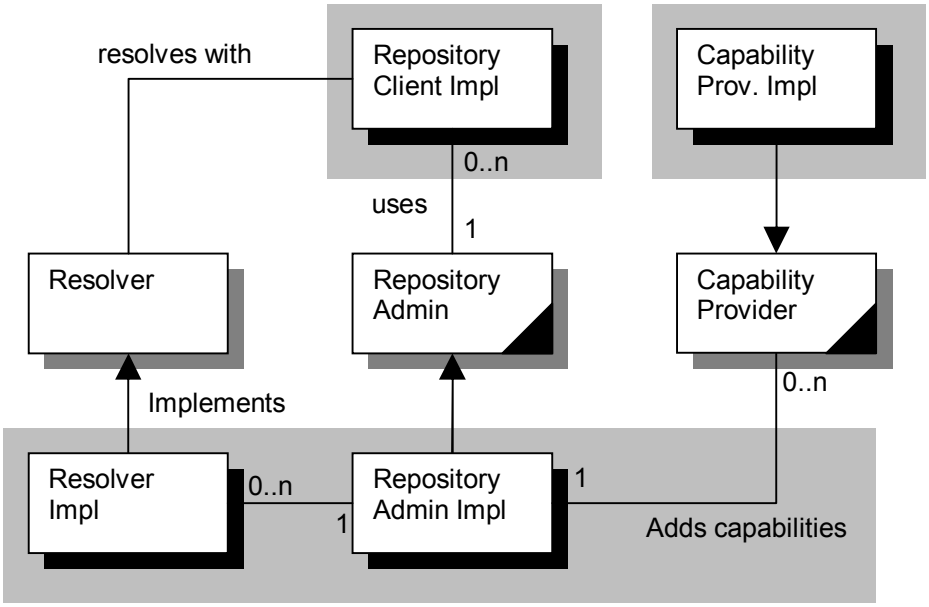
- Repository Admin – A service that provides access to a federation of repositories.
- Repository – Provides access to a set of resources that are defines in a repository file
- Resource – A description of a bundle or other artifact that can be installed on a device. A resource provides capabilities and requires capabilities of other resources or the environment.
- Capability – A named set of properties
- Requirement – An assertion on a resource's capabilities.
- Extend – A resource can act as an extension to another resource.
- Resolver – An object that can be used to find dependent and extension resources, as well as install them.
- Repository File – An XML file that can be referenced by a URL. The content contains meta data of resources and referrals to other repository files. It can be a static file or generated by a server.

All Page Within This Box

5.1.1.1 Domain Object Model



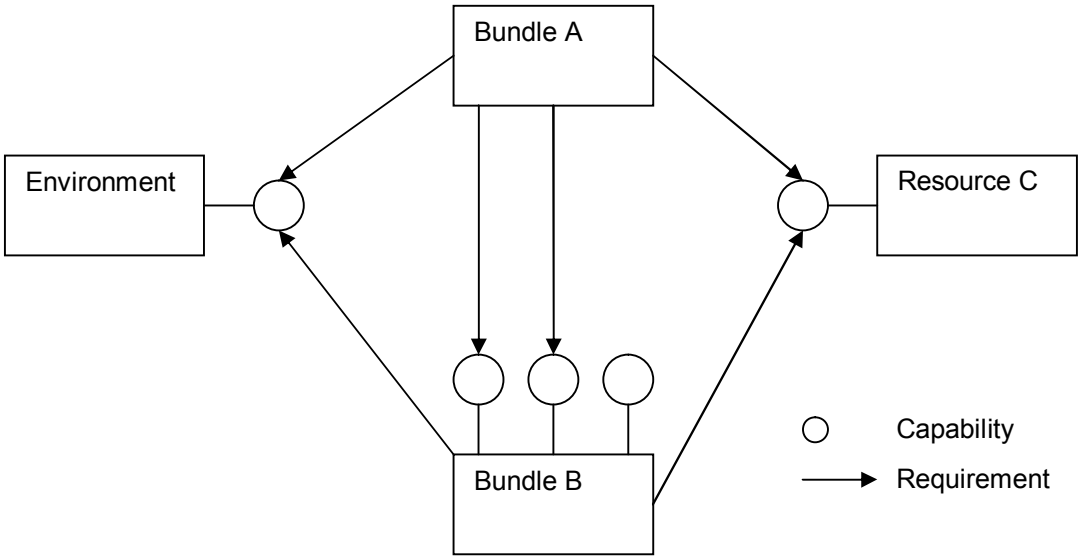
5.1.1.2 Service Model



5.2 Overview

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

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



All Page Within This Box

5.2.1 Resource

The resource is identified by the following methods:

- getName – A name for the resource that is globally unique for the function of the resource. There can exists multiple resources with the same name but a different version. Two resources with the same name and version are considered to be identical. For a bundle, this is normally mapped to the Bundle-SymbolicName manifest header.
- getVersion – A version for the resource. This must be a version usable by the OSGi Framework version class. For a bundle this is mapped to the Bundle-Version manifest header.
- getId – A local repository admin which is a handle to the resource object. This can not be used as a persistent id. Use name + version + repository URL for this.

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

1. id – The id of the resource. This id is also available from the getId method and is automatically managed by the repository; it is not possible to override this property.
2. version – The version of the resource, managed by the repository; it is not possible to override this property.
3. name – The name of the resource. Managed by the repository; it is not possible to override this property.
4. license – A URI to the license file. This element is derived from the Bundle-License manifest header.
5. description – A textual description of the bundle. This must be unformatted text. This element is derived from the Bundle-Description manifest header.
6. documentation – A URI to the documentation. This element is derived from the Bundle-DocURL manifest header.
7. copyright – A copyright statement. This element is derived from the Bundle-Copyright manifest header.
8. source – A URI to a source distribution of the resource. This element is derived from the Bundle-Source manifest header.
9. size – The size of the resource in bytes.

Property names must follow the rules for bundle symbolic names.

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

- *string* – Java String object
- *version* – org.osgi.framework.Version
- *uri* – Java URI object
- *long* –

All Page Within This Box

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

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

5.2.2 Capabilties

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

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

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

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

5.2.3 Requirements

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

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

All Page Within This Box

optionally contain a *reason*. A reason is a short description that is applicable when a requirement is the cause for the selection of a resource.

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

Requirements can be optional and/or multiple.

Optional	Multiple	Cardinality	Description
False	False	1..1	One and only one solution is required.
False	True	1..*	At least one solution is required, but multiple solutions are useful
True	False	0..1	Optional, zero or one solution is required
True	True	0..*	Optional, but multiple solutions are useful

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

5.2.4 Extends

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

5.3 Repository Admin

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

5.3.1 Discovery

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

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

5.3.2 Resolving

Resolving can be an iterative process that takes as input a set of bundles and delivers a set of required bundles and a set of optional bundles. A special Resolver object is used to simplify the API significantly. The resolving process is further described in 5.4.

The resolver method creates a new resolver.

All Page Within This Box

5.3.3 Admin

The maintenance of set of included repositories is handled by the following methods:

- addRepository – Add a new repository from a URL. This method will read the repository and any referred repositories.
- removeRepository – Remove a repository and all its referred repositories.
- listRepositories – Provide an array of Repository objects. These are only the top level repositories, referred repositories are not visible.

5.4 Resolving

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

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

resolver api

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

5.5 XML Schema

5.5.1 Namespace

The XML namespace is:

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

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

5.5.2 The XML Structure

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

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

5.5.3 Repository

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

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

All Page Within This Box

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

The repository element can only contain referral and resource elements.

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

5.5.4 Referral

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

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

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

For example:

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

5.5.5 Resource

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

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

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

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

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

For example:

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

5.5.6 Category

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

- *id* – The id of the category.

All Page Within This Box

For example:

<category id="osgi"/>

<category id="test"/>

5.5.7 Require

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

- *name* – The name of the requirement. The filter must only be applied to capabilities that have the same name.
- *filter* – The filter expression. The syntax must follow the OSGi filter syntax. The filter must correctly compare versions.
- *multiple* – If this requirements selects more than one candidate, then this is useful. The value is true or false.
- *optional* – If this requirement is necessary to satisfy the resource. The value is true or false.

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

For example:

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

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

5.5.8 Extend

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

For example:

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

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

All Page Within This Box

5.5.9 Capability

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

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

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

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

The following example shows a package export:

```
<capability name='package'>
  <p v='org.eclipse.core.internal.resources' n='package'/>
  <p v='0.0.0' t='version' n='version'/>
  <p v='true' n='x-internal:'/>
</capability>
```

5.6 Filter Extensions

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

5.6.1 Greater and Less Operators

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

5.6.2 Set Arithmetic

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

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

All Page Within This Box

```
                                                test.
```

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

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

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

5.6.3 Version Ranges

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

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

5.7 Sample XML File

```
<repository name='Untitled' time='20051210072623.031'>
  <resource version='3.0.0' name='org.osgi.test.cases.tracker'
     uri='org.osgi.test.cases.tracker-3.0.0.jar'>
    <size>
      44405
    </size>
    <documentation>
      http://www.osgi.org/
    </documentation>
    <copyright>
      Copyright (c) OSGi Alliance (2000, 2005). All Rights Reserved.
    </copyright>
    <category id='osgi'/>
    <category id='test'/>
    <capability name='bundle'>
      <p v='1' n='manifestversion'/>
      <p v='org.osgi.test.cases.tracker' n='symbolicname'/>
      <p v='3.0.0' t='version' n='version'/>
    </capability>
    <capability name='package'>
      <p v='org.osgi.test.cases.tracker' n='package'/>
      <p v='0.0.0' t='version' n='version'/>
    </capability>
    <require optional='false'  multiple='false'  name='package'
        filter='(&amp;(package=org.osgi.test.cases.util)(version&gt;=1.1.0))'>
      Import package org.osgi.test.cases.util ;version=1.1.0
    </require>
 </resource>
</repository>
```

All Page Within This Box

5.8 Querying a Web Service Based Repository

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

- keywords – A space separated (before URL encoding) list of keywords. This command must return all resources that match a keyword in the description, category, copyright, etc, case insensitive.
- requirement – A structured field. The first part is the name of the requirement, followed by a legal filter expression.
- category – A category

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

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

For example

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

5.9 Bundle Manifest Header Mapping

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

5.9.1 Bundle

Every bundle must include a 'bundle' capability with the following properties:

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

Example:

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

All Page Within This Box

5.9.2 Import and Export Package: 'package'

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

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

For example:

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

An Import-Package clause is mapped to a Requirement.

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

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

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

5.9.3 Require-Bundle

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

All Page Within This Box

- Type is 'bundle'

Assert:

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

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

For example:

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

5.9.4 Fragment-Host

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

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

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

5.9.5 Import- and Export-Service

The Import-Service and Export-Service are deprecated, however, they are still useful for management purposes. Therefore, they are mapped to the generic requirement model.

Export-Service is mapped to a capability with the name 'service'. The following properties are used.

- service – name of the service interface

For example

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

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

- service – Name of the service

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

All Page Within This Box

5.9.6 Declarative Services

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

TBD.

5.9.7 Bundle-ExecutionEnvironment

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

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

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

This requirement is UNARY.

6 Open Issues

6.1 Uses Constraint

The current specification does not address the *uses* directive on exported packages. The lack of handling this constraint makes it theoretically possible that a set of bundles is found that is resolved by the Repository resolver but can not be resolved by the Framework resolver. The following case demonstrates such a case:

```
A: import p;version=1,q;version=1
B: export p;version=1;uses:=q
   import q;version=2
C: export q;version=1
D: export q;version=2
```

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

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

6.2 Query protocol

Richard thinks the query protocol is not necessary

6.3 Licensing

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

All Page Within This Box

7 Java Documentation

skip-navbar_top
Package Class Deprecated Help
PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES All Classes

7.1 Package org.osgi.servicex.obr

Interface Summary	
Capability	A named set of properties representing some capability that is provided by its owner.
CapabilityProvider	This service interface allows third parties to provide capabilities that are present on the system but not encoded in the bundle's manifests.
Repository	Represents a repository.
RepositoryAdmin	Provides centralized access to the distributed repository.
Requirement	A named requirement specifies the need for certain capabilities with the same name.
Resolver	
Resource	A resource is an abstraction of a downloadable thing, like a bundle.

Class Summary	
RepositoryPermission	TODO Implement

skip-navbar_bottom
Package Class Deprecated Help
PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES All Classes
skip-navbar_top
Package Class Deprecated Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

7.2 org.osgi.servicex.obr
Interface Resource

public interface **Resource**

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

All Page Within This Box

Version:
$Revision: 1.5 $

Field Summary

static java.lang.String	**COPYRIGHT**
static java.lang.String	**DESCRIPTION**
static java.lang.String	**DOCUMENTATION_URI**
static java.lang.String	**ID**
static java.lang.String[]	**KEYS**
static java.lang.String	**LICENSE_URI**
static java.lang.String	**PRESENTATION_NAME**
static java.lang.String	**SIZE**
static java.lang.String	**SOURCE_URI**
static java.lang.String	**SYMBOLIC_NAME**
static java.lang.String	**URI**
static java.lang.String	**VERSION**

Method Summary

Capability[]	**getCapabilities**()
java.lang.String[]	**getCategories**()
Requirement[]	**getExtends**()
java.lang.String	**getId**()
java.lang.String	**getPresentationName**()
java.util.Map	**getProperties**()

All Page Within This Box

[Repository](#)	**getRepository**()
[Requirement](#)[]	**getRequests**()
[Requirement](#)[]	**getRequirements**()
java.lang.String	**getSymbolicName**()
java.net.URI	**getURI**()
org.osgi.framework.Version	**getVersion**()

Field Detail

7.2.1 LICENSE_URI

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

7.2.2 DESCRIPTION

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

7.2.3 DOCUMENTATION_URI

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

7.2.4 COPYRIGHT

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

7.2.5 SOURCE_URI

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

7.2.6 SYMBOLIC_NAME

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

7.2.7 PRESENTATION_NAME

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

7.2.8 ID

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

7.2.9 VERSION

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

7.2.10 URI

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

7.2.11 SIZE

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

7.2.12 KEYS

```
public static final java.lang.String[] KEYS
```

Method Detail

7.2.13 getProperties

```
public java.util.Map getProperties()
```

All Page Within This Box

7.2.14 getSymbolicName

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

7.2.15 getPresentationName

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

7.2.16 getVersion

```
public org.osgi.framework.Version getVersion()
```

7.2.17 getId

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

7.2.18 getURI

```
public java.net.URI getURI()
```

7.2.19 getRequirements

```
public Requirement[] getRequirements()
```

7.2.20 getRequests

```
public Requirement[] getRequests()
```

7.2.21 getExtends

```
public Requirement[] getExtends()
```

7.2.22 getCapabilities

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

7.2.23 getCategories

```
public java.lang.String[] getCategories()
```

7.2.24 getRepository

```
public Repository getRepository()
```

---

skip-navbar_bottom

Package Class Deprecated Help

PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

skip-navbar top

Package Class Deprecated Help

PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

7.3 org.osgi.servicex.obr
Interface Resolver

public interface **Resolver**

Method Summary

void	**add**(Resource resource)
void	**deploy**()
Resource[]	**getAddedResources**()
Resource[]	**getOptionalResources**()
Requirement[]	**getReason**(Resource resource)
Resource[]	**getRequiredResources**()
Resource[]	**getResources**(Requirement requiremen)
Requirement[]	**getUnsatisfiedRequirements**()
boolean	**resolve**()

Method Detail

7.3.1 add

public void **add**(Resource resource)

7.3.2 getUnsatisfiedRequirements

All Page Within This Box

public Requirement[] **getUnsatisfiedRequirements**()

7.3.3 getOptionalResources

public Resource[] **getOptionalResources**()

7.3.4 getReason

public Requirement[] **getReason**(Resource resource)

7.3.5 getResources

public Resource[] **getResources**(Requirement requiremen)

7.3.6 getRequiredResources

public Resource[] **getRequiredResources**()

7.3.7 getAddedResources

public Resource[] **getAddedResources**()

7.3.8 resolve

public boolean **resolve**()

7.3.9 deploy

public void **deploy**()

skip-navbar_bottom
Package Class Deprecated Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Deprecated Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

7.4 org.osgi.servicex.obr
Interface Requirement

public interface **Requirement**

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

Version:
$Revision: 1.5 $

Method Summary

java.lang.String	**getComment**()
java.lang.String	**getFilter**() Return the filter.
java.lang.String	**getName**() Return the name of the requirement.
boolean	**isMultiple**()
boolean	**isOptional**()
boolean	**isSatisfied**(Capability capability)

Method Detail

7.4.1 getName

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

7.4.2 getFilter

```
public java.lang.String getFilter()
```
Return the filter.

7.4.3 isMultiple

```
public boolean isMultiple()
```

7.4.4 isOptional

```
public boolean isOptional()
```

7.4.5 getComment

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

All Page Within This Box

7.4.6 isSatisfied

```
public boolean isSatisfied(Capability capability)
```

skip-navbar_bottom
Package Class Deprecated Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Deprecated Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

7.5 org.osgi.servicex.obr
Class RepositoryPermission

```
java.lang.Object
  └java.security.Permission
      └java.security.BasicPermission
          └org.osgi.servicex.obr.RepositoryPermission
```

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

public class **RepositoryPermission**
extends java.security.BasicPermission

TODO Implement

Version:
$Revision: 1.1 $

See Also:
Serialized Form

Constructor Summary

RepositoryPermission(java.lang.String name)

Methods inherited from class java.security.BasicPermission
equals, getActions, hashCode, implies, newPermissionCollection

Methods inherited from class java.security.Permission
checkGuard, getName, toString

All Page Within This Box

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

7.5.1 RepositoryPermission

```
public RepositoryPermission(java.lang.String name)
```

skip-navbar_bottom
Package Class Deprecated Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Deprecated Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

7.6 org.osgi.servicex.obr
Interface RepositoryAdmin

public interface **RepositoryAdmin**

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

1. Capabilities - Capabilities provide a named aspect: a bundle, a display, memory, etc.
2. Requirements - A named filter expression. The filter must be satisfied by one or more Capabilties with the given name. These capabilities can come from other resources or from the platform. If multiple resources provide the requested capability, one is selected. (### what algorithm? ###)
3. Requests - Requests are like requirements, except that a request can be fullfilled by 0..n resources. This feature can be used to link to resources that are compatible with the given resource and provide extra functionality. For example, a bundle could request all its known fragments. The UI associated with the repository could list these as optional downloads.

Version:
$Revision: 1.2 $

Method Summary

Repository	**addRepository**(java.net.URL repository) Add a new repository to the federation.
Resource[]	**discoverResources**(java.lang.String filterExpr) Discover any resources that match the given filter.

Resource	**getResource**(java.lang.String respositoryId)
Repository[]	**listRepositories**() List all the repositories.
boolean	**removeRepository**(java.net.URL repository)
Resolver	**resolver**() Create a resolver.

Method Detail

7.6.1 discoverResources

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

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

Parameters:
filterExpr - A standard OSGi filter
Returns:
List of resources matching the filters.

7.6.2 resolver

public Resolver **resolver**()
Create a resolver.
Returns:

7.6.3 addRepository

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

All Page Within This Box

repository -
Returns:
Throws:
java.lang.Exception

7.6.4 removeRepository

public boolean **removeRepository**(java.net.URL repository)

7.6.5 listRepositories

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

7.6.6 getResource

public Resource **getResource**(java.lang.String respositoryId)

skip-navbar_bottom
Package Class Deprecated Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Deprecated Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

7.7 org.osgi.servicex.obr
Interface Repository

public interface **Repository**

Represents a repository.

Version:
$Revision: 1.2 $

Method Summary

long	**getLastModified**()
java.lang.String	**getName**() Return the name of this repository.
Resource[]	**getResources**() Return the resources for this repository.

All Page Within This Box

java.net.URL	**getURL**() Return the associated URL for the repository.

Method Detail

7.7.1 getURL

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

Return the associated URL for the repository.

7.7.2 getResources

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

Return the resources for this repository.

7.7.3 getName

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

Return the name of this reposotory.

Returns:

a non-null name

7.7.4 getLastModified

`public long` **`getLastModified`**`()`

---

skip-navbar_bottom

Package Class Deprecated Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top

Package Class Deprecated Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

7.8 org.osgi.servicex.obr
Interface CapabilityProvider

---

public interface **CapabilityProvider**

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

1. A Set of certificates

All Page Within This Box

2. Dimensions of the screen
3. Amount of memory
4. ...

Version:
$Revision: 1.1 $

Method Summary

Capability[]	**getCapabilities**() Return a set of capabilities.

Method Detail

7.8.1 getCapabilities

public Capability[] **getCapabilities**()

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

Returns:
Set of capabilities

skip-navbar_bottom
Package Class Deprecated Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Deprecated Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

7.9 org.osgi.servicex.obr
Interface Capability

public interface **Capability**

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

Version:
$Revision: 1.2 $

Method Summary

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

java.util.Map	**getProperties**() Return the set of properties.

Method Detail

7.9.1 getName

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

Return the name of the capability.

7.9.2 getProperties

`public java.util.Map` **`getProperties`**`()`

Return the set of properties. Notice that the value of the properties is a list of values.

Returns:

a Map

---

skip-navbar_bottom

Package **Class** **Deprecated** **Help**

PREV CLASS **NEXT CLASS** | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

---

8 Security Considerations

8.1 Repository Permission

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

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

The name of this permission is irrelevant.

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

[3]. JSR 124 J2EE Client Provisioning http://www.jcp.org/en/jsr/detail?id=124

9.2 Author's Address

Name	Peter Kriens
Company	aQute
Address	9c, Avenue St. Drezery, Beaulieu, France
Voice	+33 467542167
e-mail	Peter.Kriens@aQute.biz
Name	Richard S. Hall
Company	
Address	Saginaw, Mi, USA
Voice	+1
e-mail	Heavy@ungoverned.org

9.3 Acronyms and Abbreviations

9.4 End of Document

All Page Within This Box