# RFC 187 - Complex Repository Requirements

Final

28 Pages

## Abstract

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

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

# 0 Document Information

## 0.1 License

### DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

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

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

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

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

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

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

## 0.2 Trademarks

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

## 0.3 Feedback

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

## 0.4 Table of Contents

## 0.5 Terminology and Document Conventions

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

```
Source code is shown in this typeface.
```

## 0.6 Revision History

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

| Revision | Date | Comments |
|---|---|---|
| Initial | 09/01/12 | David Bosschaert (Red Hat) initial version |
| 0.2 | November, 2012 | David Bosschaert (Red Hat) incorporate feedback from the Basel F2F |
| 0.3 | December, 2012 | David Bosschaert (Red Hat) feedback from Orlando F2F |
| 0.4 | February, 2013 | David Bosschaert (Red Hat) feedback from Austin F2F |
| 0.5 | April, 2013 | David Bosschaert (Red Hat) add Javadoc |
| Final | 2015-01-30 | Mark final for EG vote. |

# 1 Introduction

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

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

# 2 Application Domain

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

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

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

- https://mail.osgi.org/pipermail/eeg/2011-December/008849.html

and this bug in the OSGi bug system:

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

# 3 Problem Description

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

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

```
Require-Capability: osgi.wiring.package;
  filter:="(&(osgi.wiring.package=javax.mail)(version>=1.4)(!(version>=2)))";
```

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

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

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

The OSGi R5 Requirement interface is defined as follows:

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

The OSGi R5 Repository Service interface is defined as:

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

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

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

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

# 4 Requirements

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

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

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

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

# 5 Technical Solution

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

```
public interface Repository {
  Map<Requirement, Collection<Capability>>
    findProviders(Collection<? extends Requirement> requirements);

  Collection<Resource> findProviders(RequirementExpression requirementExpression);

  ExpressionCombiner getExpressionCombiner();

  RequirementBuilder newRequirementBuilder(String namespace);
}
```

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

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

```
  Repository repo = ...; // from Service Registry
```

```
  Requirement req1 = repo.newRequirementBuilder("ns1").
    addDirective("filter", "(org.foo.ns1=val1)").
    addDirective("cardinality", "multiple").build();

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

  Requirement req3 = repo.newRequirementBuilder("ns3").build();

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

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

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

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

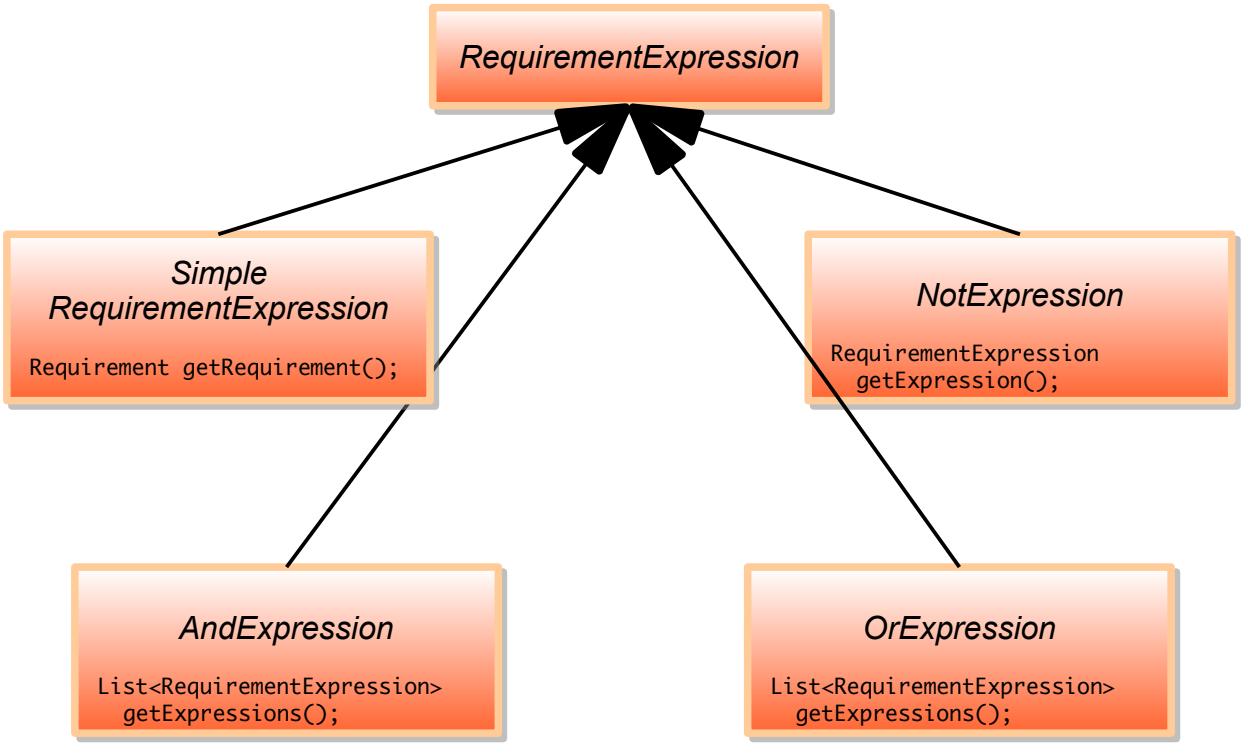
Once all the information is provided to the builder, the desired immutable Requirement object can be obtained by calling the build() method:

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

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

  Requirement build();
}
```

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

```
public interface ExpressionCombiner {
  RequirementExpression and(Requirement... reqs);
  RequirementExpression and(RequirementExpression... reqs);
  RequirementExpression expression(Requirement req);
  RequirementExpression not(Requirement req);
  RequirementExpression not(RequirementExpression req);
  RequirementExpression or(Requirement... reqs);
  RequirementExpression or(RequirementExpression... reqs);
}
```

The RequirementExpression interface hierarchy is as follows:




The interface hierarchy represents a number of holder interfaces that each can hold one or more RequirementExpression objects. The SimpleRequirementExpression can hold and existing org.osgi.resource.Requirement object, allowing it to be represented as a RequirementExpression without the need for modification.

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

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

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

```
private static String printExpression(RequirementExpression req) {
  if (req instanceof SimpleRequirementExpression) {
    return ((SimpleRequirementExpression)req).getRequirement().toString();
  } else if (req instanceof NotExpression) {
```

```
    return "not" + printExpression(((NotExpression) req).getExpression()) + "";
  } else if (req instanceof AndExpression) {
    StringBuilder sb = new StringBuilder();
    AndExpression and = (AndExpressions) req;
    for (int i = 0; i < and.getExpressions().size(); i++) {
      if (i > 0) sb.append(" and ");
      sb.append(printExpression(and.getExpressions().get(i)));
    }
    return "(" + sb.toString() + ")";
  } else if (req instanceof OrExpression) {
    StringBuilder sb = new StringBuilder();
    OrExpression or = (OrExpression) req;
    for (int i = 0; i < or.getExpressions().size(); i++) {
      if (i > 0) sb.append(" or ");
      sb.append(printExpression(or.getExpressions().get(i)));
    }
    return "(" + sb.toString() + ")";
  }
  throw new IllegalStateException();
}
```

Given the above example input, the printRequirements outputs the following string:

```
(not(ns1 and ns2) or ns3)
```

## 5.1 Notes

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

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

# 6 Data Transfer Objects

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

# 7 Javadoc

# OSGi Javadoc

4/19/13 1:11 PM

| Package Summary | | Page |
|---|---|---|
| org.osgi.service.repository | Repository Service Package Version 1.0. | 12 |

# Package org.osgi.service.repository

Repository Service Package Version 1.0.

**See:**
[Description](#)

| Interface Summary | | Page |
|---|---|---|
| ***AndExpression*** | A `RequirementExpression` representing multiple requirements combined together using the `and` operator. | *13* |
| ***ExpressionCombiner*** | An `ExpressionCombiner` can be used to combine multiple requirements into a single complex requirement using the `and`, `or` and `not` operators. | *16* |
| ***NotExpression*** | A `RequirementExpression` representing the negative of a requirement, the `not` operator. | *19* |
| ***OrExpression*** | A `RequirementExpression` representing multiple requirements combined together using the `or` operator. | *20* |
| ***Repository*** | A repository service that contains `resources`. | *21* |
| ***RepositoryContent*** | An accessor for the default content of a resource. | *23* |
| ***RequirementBuilder*** | A builder for `Requirement` objects. | *24* |
| ***RequirementExpression*** | The base interface of all Requirement Expressions. | *26* |
| ***SimpleRequirementExpression*** | A wrapper to represent a simple `org.osgi.resource.Requirement` object as a `RequirementExpression`. | *27* |

| Class Summary | | Page |
|---|---|---|
| **ContentNamespace** | Content Capability and Requirement Namespace. | *14* |

# Package org.osgi.service.repository Description

Repository Service Package Version 1.0.

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

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.service.repository; version="[1.1,2.0)"
```

Example import for providers implementing the API in this package:

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

# Interface AndExpression

**org.osgi.service.repository**

**All Superinterfaces:**
RequirementExpression

---

```
public interface AndExpression
extends RequirementExpression
```

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

**ThreadSafe**

---

| Method Summary | | Page |
|---|---|---|
| List<RequirementExpression> | **getRequirements**()<br>Obtain the requirements that are combined using the `and` operator. | 13 |

## Method Detail

### getRequirements

```
List<RequirementExpression> getRequirements()
```

Obtain the requirements that are combined using the `and` operator.

**Returns:**
The requirements, represented as `RequirementExpression` objects.

# Class ContentNamespace

org.osgi.service.repository

```
java.lang.Object
  └org.osgi.resource.Namespace
      └org.osgi.service.repository.ContentNamespace
```

---

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

Content Capability and Requirement Namespace.

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

**Immutable**

---

| Field Summary | | Page |
|---|---|---|
| static String | **CAPABILITY_MIME_ATTRIBUTE**<br>The capability attribute that defines the IANA MIME Type/Format for this content. | 15 |
| static String | **CAPABILITY_SIZE_ATTRIBUTE**<br>The capability attribute that contains the size, in bytes, of the content. | 15 |
| static String | **CAPABILITY_URL_ATTRIBUTE**<br>The capability attribute that contains the URL to the content. | 14 |
| static String | **CONTENT_NAMESPACE**<br>Namespace name for content capabilities and requirements. | 14 |

| Fields inherited from class org.osgi.resource.Namespace |
|---|
| CAPABILITY_EFFECTIVE_DIRECTIVE, CAPABILITY_USES_DIRECTIVE, CARDINALITY_MULTIPLE, CARDINALITY_SINGLE, EFFECTIVE_ACTIVE, EFFECTIVE_RESOLVE, REQUIREMENT_CARDINALITY_DIRECTIVE, REQUIREMENT_EFFECTIVE_DIRECTIVE, REQUIREMENT_FILTER_DIRECTIVE, REQUIREMENT_RESOLUTION_DIRECTIVE, RESOLUTION_MANDATORY, RESOLUTION_OPTIONAL |

## Field Detail

### CONTENT_NAMESPACE

```
public static final String CONTENT_NAMESPACE = "osgi.content"
```

Namespace name for content capabilities and requirements.

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

---

### CAPABILITY_URL_ATTRIBUTE

```
public static final String CAPABILITY_URL_ATTRIBUTE = "url"
```

The capability attribute that contains the URL to the content.

---

## CAPABILITY_SIZE_ATTRIBUTE

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

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

---

## CAPABILITY_MIME_ATTRIBUTE

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

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

# Interface ExpressionCombiner

org.osgi.service.repository

---

`public interface` **`ExpressionCombiner`**

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

---

| Method Summary | | Page |
|---|---|---|
| RequirementExpression | **and**(org.osgi.resource.Requirement... reqs)<br>Combine multiple `org.osgi.resource.Requirement` objects into a single expression using the `and` operator. | 16 |
| RequirementExpression | **and**(RequirementExpression... reqs)<br>Combine multiple `RequirementExpression` objects into a single expression using the `and` operator. | 16 |
| RequirementExpression | **expression**(org.osgi.resource.Requirement req)<br>Convert a `org.osgi.resource.Requirement` into a `RequirementExpression`. | 17 |
| RequirementExpression | **not**(org.osgi.resource.Requirement req)<br>Provide the negative of a `org.osgi.resource.Requirement`. | 17 |
| RequirementExpression | **not**(RequirementExpression req)<br>Provide the negative of a `RequirementExpression`. | 17 |
| RequirementExpression | **or**(org.osgi.resource.Requirement... reqs)<br>Combine multiple `org.osgi.resource.Requirement` objects into a single expression using the `or` operator. | 17 |
| RequirementExpression | **or**(RequirementExpression... reqs)<br>Combine multiple `RequirementExpression` objects into a single expression using the `or` operator. | 17 |

## Method Detail

### and

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

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

**Parameters:**
`reqs` - The requirements to combine.
**Returns:**
A `RequirementExpression` representing the combined requirements.

---

### and

`RequirementExpression` **`and`**`(RequirementExpression... reqs)`

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

**Parameters:**
`reqs` - The requirements to combine.
**Returns:**
A `RequirementExpression` representing the combined requirements.

## expression

RequirementExpression **expression**(org.osgi.resource.Requirement req)

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

**Parameters:**
req - The requirement to convert.
**Returns:**
A RequirementExpression representing the requirement.

## not

RequirementExpression **not**(org.osgi.resource.Requirement req)

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

**Parameters:**
req - The requirement to provide the negative of.
**Returns:**
A RequirementExpression representing the negative of the requirement.

## not

RequirementExpression **not**(RequirementExpression req)

Provide the negative of a RequirementExpression.

**Parameters:**
req - The requirement to provide the negative of.
**Returns:**
A RequirementExpression representing the negative of the requirement.

## or

RequirementExpression **or**(org.osgi.resource.Requirement... reqs)

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

**Parameters:**
reqs - The requirements to combine.
**Returns:**
A RequirementExpression representing the combined requirements.

## or

RequirementExpression **or**(RequirementExpression... reqs)

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

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

**Returns:**

A RequirementExpression representing the combined requirements.

# Interface NotExpression

**org.osgi.service.repository**

**All Superinterfaces:**
RequirementExpression

---

```
public interface NotExpression
extends RequirementExpression
```

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

**ThreadSafe**

---

| Method Summary | Page |
|---|---|
| RequirementExpression | **getRequirement**()<br>Obtain the requirement that is negated. | 19 |

## Method Detail

### getRequirement

`RequirementExpression` **getRequirement**()

Obtain the requirement that is negated.

**Returns:**
The requirement, represented as `RequirementExpression`.

# Interface OrExpression

org.osgi.service.repository

**All Superinterfaces:**
RequirementExpression

---

```
public interface OrExpression
extends RequirementExpression
```

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

**ThreadSafe**

---

| Method Summary | | Page |
|---|---|---|
| List<RequirementExpression> | **getRequirements**()<br>Obtain the requirements that are combined using the or operator. | 20 |

## Method Detail

### getRequirements

```
List<RequirementExpression> getRequirements()
```

Obtain the requirements that are combined using the or operator.

**Returns:**
The requirements, represented as RequirementExpression objects.

# Interface Repository

org.osgi.service.repository

---

```
public interface Repository
```

A repository service that contains `resources`.

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

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

**ThreadSafe**

---

| Field Summary | | Page |
|---|---|---|
| String | **URL**<br>Service property to provide URLs related to this repository. | 21 |

| Method Summary | | Page |
|---|---|---|
| Map<org.osgi.resource.Requirement,Collection<org.osgi.resource.Capability>> | **findProviders**(Collection<? extends org.osgi.resource.Requirement> requirements)<br>Find the capabilities that match the specified requirements. | 21 |
| Collection<org.osgi.resource.Resource> | **findProviders**(RequirementExpression requirementExpression)<br>Find the resources that match the specified `RequirementExpression` | 22 |
| ExpressionCombiner | **getExpressionCombiner**()<br>Obtain an `ExpressionCombiner` implementation. | 22 |
| RequirementBuilder | **newRequirementBuilder**(String namespace)<br>Obtain a `RequirementBuilder` implementation which provides a convenient way to create a requirement. | 22 |

## Field Detail

### URL

```
public static final String URL = "repository.url"
```

Service property to provide URLs related to this repository.

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

## Method Detail

### findProviders

```
Map<org.osgi.resource.Requirement,Collection<org.osgi.resource.Capability>> findProviders(Collection<? extends org.osgi.resource.Requirement> requirements)
```

Find the capabilities that match the specified requirements.

**Parameters:**
`requirements` - The requirements for which matching capabilities should be returned. Must not be `null`.
**Returns:**
A map of matching capabilities for the specified requirements. Each specified requirement must appear as a key in the map. If there are no matching capabilities for a specified requirement, then the value in the map for the specified requirement must be an empty collection. The returned map is the property of the caller and can be modified by the caller.

---

## findProviders

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

Find the resources that match the specified `RequirementExpression`

**Parameters:**
`requirementExpression` - The `RequirementExpression` for which matching capabilities should be returned. Must not be `null` .
**Returns:**
A collection of matching `Resource`s. If there are no matching resources, an empty collection is returned.

---

## getExpressionCombiner

`ExpressionCombiner` **`getExpressionCombiner`**`()`

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

**Returns:**
An `ExpressionCombiner`.

---

## newRequirementBuilder

`RequirementBuilder` **`newRequirementBuilder`**`(String namespace)`

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

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

**Parameters:**
`namespace` - The namespace for the requirement to be constructed.
**Returns:**
A requirement builder for a requirement in the given namespace.

# Interface RepositoryContent

org.osgi.service.repository

---

```
public interface RepositoryContent
```

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

**ThreadSafe**

---

| Method Summary | | Page |
|---|---|---|
| InputStream | **getContent**()<br>Returns a new input stream to the default format of this resource. | 23 |

## Method Detail

### getContent

```
InputStream getContent()
```

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

**Returns:**
A new input stream for associated resource.

# Interface RequirementBuilder

org.osgi.service.repository

public interface **RequirementBuilder**

A builder for `Requirement` objects.

| Method Summary | | Page |
|---|---|---|
| RequirementBuilder | **addAttribute**(String name, Object value)<br>Add an attribute to the requirement. | 24 |
| RequirementBuilder | **addDirective**(String name, String value)<br>Add a directive to the requirement. | 24 |
| org.osgi.resource.Requirement | **build**()<br>Build the requirement according to the specification provided to the builder. | 25 |
| RequirementBuilder | **setAttributes**(Map<String,Object> attrs)<br>Set all the attributes to the values in the provided map. | 24 |
| RequirementBuilder | **setDirectives**(Map<String,String> dirs)<br>Set all the directives to the values in the provided map. | 25 |
| RequirementBuilder | **setResource**(org.osgi.resource.Resource resource)<br>Specifies the `Resource` object for the requirement. | 25 |

## Method Detail

### addAttribute

RequirementBuilder **addAttribute**(String name,
Object value)

Add an attribute to the requirement.

**Parameters:**
`name` - The attribute name.
`value` - The attribute value.
**Returns:**
a builder object that can be used to further define the requirement.

### addDirective

RequirementBuilder **addDirective**(String name,
String value)

Add a directive to the requirement.

**Parameters:**
`name` - The directive name.
`value` - The directive value.
**Returns:**
a builder object that can be used to further define the requirement.

### setAttributes

RequirementBuilder **setAttributes**(Map<String,Object> attrs)

Set all the attributes to the values in the provided map. This will replace any previous attribute set on the builder.

**Parameters:**
`attrs` - The map of attributes to use.
**Returns:**
a builder object that can be used to further define the requirement.

---

## setDirectives

`RequirementBuilder` **`setDirectives`**`(Map<String,String> dirs)`

Set all the directives to the values in the provided map. This will replace any previous directives set on the builder.

**Parameters:**
`dirs` - The map of directives to use.
**Returns:**
a builder object that can be used to further define the requirement.

---

## setResource

`RequirementBuilder` **`setResource`**`(org.osgi.resource.Resource resource)`

Specifies the `Resource` object for the requirement. Note that providing a resource is optional.

**Parameters:**
`resource` - The resource for the requirement. Will overwrite any previous resource if provided.
**Returns:**
a builder object that can be used to further define the requirement.

---

## build

`org.osgi.resource.Requirement` **`build`**`()`

Build the requirement according to the specification provided to the builder.

**Returns:**
the requirement.

# Interface RequirementExpression

**org.osgi.service.repository**

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

---

```
public interface RequirementExpression
```

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

**ThreadSafe**

# Interface SimpleRequirementExpression

org.osgi.service.repository

**All Superinterfaces:**
RequirementExpression

---

```
public interface SimpleRequirementExpression
extends RequirementExpression
```

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

**ThreadSafe**

---

| Method Summary | Page |
|---|---|
| `org.osgi.resource.Requirement` | **`getRequirement`**`()` Obtain the wrapped `org.osgi.resource.Requirement` object. | *27* |

## Method Detail

### getRequirement

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

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

**Returns:**
The wrapped object.



# 8 Considered Alternatives

*For posterity, record the design alternatives that were considered but rejected along with the reason for rejection. This is especially important for external/earlier solutions that were deemed not applicable.*

# 9 Security Considerations

No change from the OSGi Repository 1.0 specification.

# 10 Document Support

## 10.1 References

[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

## 10.2 Author's Address



| | |
|---|---|
| Name | David Bosschaert |
| Company | Red Hat |
| Address | |
| Voice | |
| e-mail | david@redhat.com |

| | |
|---|---|
| Name | |
| Company | |
| Address | |
| Voice | |
| e-mail | |



## 10.3 Acronyms and Abbreviations

## 10.4 End of Document