RFC 241 – Features

Draft

17 Pages

Abstract

OSGi is regularly used as a platform for running applications comprised of a large number of bundles, configurations and other artifacts. However it is lacking a developer friendly mechanism to define such applications. This RFC aims at describing a technical solution to address this challenge.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

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

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

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

Table 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 10.1.

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

0.6 Revision History

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

Revision	Date	Comments
Initial	February 2019	David Bosschaert, initial version.
0.1	May 2019	David Bosschaert, feedback from the Berlin F2F

1 Introduction

OSGi has become a platform capable of running large applications for a variety of purposes, including rich client applications, server-side systems and cloud and container based architectures. As these applications are generally based on many bundles, describing each bundle individually in the application definition becomes unwieldy once the number of bundles reaches a certain level.

Furthermore, OSGi has no mechanism to describe other elements of the application definition, such as configuration or custom artifacts.

The requirements for a higher level to describe OSGi applications that encapsulates the details of the various components that the application is built up from are described in RFP 188. They are also available in this document. This RFC aims to describe the technical solution for the requirements identified in RFP 188. It allows the description of an entire OSGi-based application based on reusable components and includes everything related to this application, including configuration, framework properties, capabilities, requirements and custom artifacts.

2 Application Domain

When developing large enterprise applications it is often the case that very few people know the role of every bundle or configuration item in the application. To keep the architecture understandable a grouping mechanism is needed that allows for the representation of parts of the application into larger entities that keep reasoning about the application manageable. In such a domain members of teams spread across the organization will need to be able to both develop new parts for the application as well as make tweaks or enhancements to their respective parts such as adding configuration and resources or changing one or more bundles relevant to their part of the application.

The higher level constructs that define the application should be reusable in different contents, for example if one team has developed a component to handle job processing, different applications should be able to use it, and if needed tune its configuration or other aspects so that it works in each setting without having to know each and every detail, bundle etc that the job processing component is built up from.

This RFP aims solving the problem of defining (large) applications in OSGi in a way that’s easy for humans and teams.

2.1 Relation to existing OSGi specifications

2.1.1 Subsystems Specification

While some might say that subsystems were designed for the purposes outlined in this RFP, subsystems are rather a possible way to implement the runtime realization of some aspects of the features. Subsystems are lacking authoring support and don't provide an architect-friendly design-time source format. Additionally, subsystems are limited to bundles, features often additionally declare configuration, custom content and custom metadata. Experience has shown that while subsystems work, authors of large systems find it difficult to work directly with these.

2.1.2 Deployment Admin Specification

The Deployment Admin specification also defines a deployable application format. These deployables are somewhat limited in that multiple deployment admin applications cannot have overlapping bundles, making this specification not very useful as many applications share certain dependencies. Additionally, the Deployment Admin specification does not define a format to architect features.

2.1.3 Application Admin Specification

The Application Admin Specification allows the deployment and management of Applications in OSGi. This specification is primarily aimed at UI-based applications. While this application provides a run-time API for deployment and management of applications, it does not provide a way to model features and applications for a systems architect.

2.2 Relation to existing Open Source solutions

A number of existing solutions exist both in Open Source as well as in closed source. From the Open Source space Apache Karaf Features are popular, as well as Eclipse Features. Additionally Apache Felix Bundle Archives provide a mechanism that could be used to deploy features.

Apache Sling Features provide a way to design and run features using JSON.

Bnd provides a mechanism to create an application runfile from a set of seed bundles, matching requirements against capabilities provided through one or more repositories.

Knowledge of the existing solutions is used to influence the requirements in this document.

2.3 Roles

The following section outlines roles involved in the creation of Feature-based OSGi applications. Note that different roles may be performed by the same individual.

Bundle Developer – A Bundle Developer writes OSGi bundle code. The Bundle Developer typically has a small scope and focuses on individual bundles or a small number of bundles that provide a cohesive piece of functionality.

Feature Developer – A Feature Developer creates OSGi features by collecting multiple bundles together to create higher level components.

Application Architect – The Application Architect designs a product by putting a number of high-level components together in a document. She defines the interaction between components in the product, and the external interactions of the product.

Application Assembler – The Application Assembler takes the input from the Application Architect and maps it to available features and configuration. He creates a high-level feature representing the application from existing features with added configuration.

Application Deployer/Administrator – The Application Deployer takes the feature created by the Application Assembler and turns it into a runnable application. He does this by mapping all requirements to capabilities and by resolving all version ranges to a specific version. He sets configuration to integrate with external systems such as databases, external microservices and others. He then runs the application on his infrastructure.

Quality Engineer – A Quality Engineer needs to test an application before it's released. The QE may also be asked to ensure that the application still works when one or more individual bundles or features, configuration or other resources are replaced with different ones.

2.4 Terminology + Abbreviations

Feature – A feature combines a number of bundles together to provide a logical piece of functionality. Features may also depend on other features, configuration and other artifacts.

Complete Feature – A complete feature has no unresolved dependencies and has all required configuration provided. A complete feature is still a regular feature and can be used everywhere a regular feature can. However some tools or scenarios may require complete features; these cannot operate on features that are not complete.

Feature Deployment Agent – An entity which is capable of consuming feature models, to produce a representation for a runtime context.

3 Problem Description

OSGi has no support for describing large applications. Application developers need to come up with their own way to do this. When applications are getting larger and are developed by multiple teams this becomes a challenge, especially in cases where the application is composed of multiple features each of which are groups of bundles, configuration, metadata and other artifacts.

4 Requirements

This specification should meet the following requirements:

- FM010 – The feature model should be described through a text format which is easily consumable by both humans and machines, that can be edited with common editors and support text-based diff operations.
- FM020 – A feature must be described through a single file.
- FM040 – The feature model language must support comments.
- FM050 – The feature model may support more than one text-based definition language where the language used can be easily inferred, for example from the file extension.
- FM060 – The feature model should provide support for long and multi-line values without creating files that become hard to handle.
- FM070 – A feature must have a version.
- FM080 – A feature must have a unique identifier, which contains the version.
- FM090 – A feature identifier must be mappable to Apache Maven coordinates.
- FM100 – It must be possible to specify the bundles belonging to the feature, including version.
- FM111 – It must be possible to identify a bundle using repository coordinates, for example for a Maven repository.
- FM120 – The feature model must allow the specification of the order in which the bundles inside the feature are started. This should be relative to when the feature itself is started.
- FM130 – It must be possible to define whether a bundle is always enabled in a feature or conditionally enabled.
- FM140 – It must be possible to associate any additional metadata like a hash with a bundle.
- FM150 – It must be possible to specify the OSGi configurations for a feature.
- FM160 – Both normal OSGi configurations as well as factory configurations must be supported. The feature model must support all data types supported by the OSGi Configuration Admin specification.
- FM170 – The OSGi configuration resource format as defined in the OSGi Configurator Specification must be supported.
- FM180 – It must be possible to associate an OSGi configuration with a bundle within a feature. If the bundle is not enabled then the associated configuration also does not get installed.
- FM190 – It must be possible to define framework launch properties in a feature.
- FM195 – it must be possibe to define system properties in a feature.
- FM200 – The feature model must be extensible to allow other artifacts than bundles.
- FM211 – It must be possible to identify artifacts in a feature using repository coordinates, for example for a Maven repository.

- FM220 – It must be possible to associate any additional metadata like a hash with an artifact.
- FM230 – It must be possible to define whether an artifact always enabled in a feature or conditionally enabled.
- FM260 – A feature must be able to specify additional requirements and capabilities that extend the requirements and capabilities from the contained artifacts.
- FM270 – A feature must be able to use another feature as a prototype.
- FM280 – A feature must be able to depend on other features through the requirements/capabilities model based on the feature contents. The feature model must be able to deal with circular dependencies. However, there must be no way of explicitly requiring a feature from another feature.
- FM290 – The feature model must describe how several features are aggregated to build a higher level feature. This description must include all parts of the feature model (bundles, configurations, framework properties etc.). The process should be general for extensions, which means it should describe how extensions are aggregated without requiring the model implementation to know the type of extension.
- FM300 – It must be possible to declare that a feature is transitively closed, this defines a ‘Complete Feature’.
- FM305 – The solution may define a packaging format for features, including their contents.
- FM310 – When features are aggregated, to create a higher level feature, and a clash is detected wrt their contents, a conflict resolution mechanism must be defined.
- FM340 – The feature model must calculate the startup order of bundles for an aggregated feature respecting the dependencies between features and their contents.
- FM350 – The feature model must support variables to be used for configurations and framework properties, avoiding the need to repeat the same value several times, and to allow late binding.
- FM400 – It must be possible to specify the framework implementation to launch as part of the feature model.
- FM430 – The feature model must support additional, optional information about the feature like a human readable title, a description, vendor and licensing information.
- FM440 – The feature model must use a semantically versioned descriptor format so that if the format evolves in the future users can state in feature model files what version they are written for.
- FM500 – All artifacts need the ability to establish trust and detect tampering, for example via signing.
- FM510 – The feature model should support conditionally including bundles, based on system properties and/or capabilities.
- FM520 – It should be possible to specify classpath, module-path and JVM/System properties in a feature file.

5 Technical Solution

5.1 Feature Model

The central concept of this specification is the Feature Model. Each Feature defined using the Feature Model has a unique ID and version, which together define the Feature identity. It can hold a number of entities, including a list of bundles, configurations, capabilities, requirements and others. Features are extensible, so a Feature can also hold any number of custom entities which are related to the Feature.

Features may have dependencies on other Features. Features inherit the capabilities and requirements from all bundles listed in the Feature, and can also have additional capabilities and requirements on the Feature level. Multiple Features can be combined together (Todor: what does it mean to combine features) to have all dependencies satisfied within a single Feature. A Feature which does not have any unsatisfied dependencies is said to be *complete* – needs additional context *(Christian: the target platform should be part of this)*. Could be outside, or expressed in an extension.

Once created a Feature is immutable. It cannot be modified. However another Feature with a different identity can be created which is based on a given Feature using the prototype mechanism.

5.1.1 Bundles

Features often list a number of bundles that provide the functionality provided by the Feature. Bundles are listed by referencing them so that they can be resolved from a repository. Bundles can have metadata associated with them, such as the relative start order of the bundle in the Feature (needs conflict resolution). Custom metadata can also be provided. A single Feature can provide multiple versions of the same bundle, if desired.

5.1.2 Configurations

Features support configuration using the OSGi Configurator syntax. ~~Listing configuration in a feature adds this configuration to the OSGi Configuration Admin service when the feature is launched.~~

PID or Factory PID conflicts need conflict resolution.

5.1.3 Framework properties

When a Feature is launched in an OSGi framework it may be necessary to specify Framework properties. These can be provided in the Framework Properties section of the Feature Model.

If the Feature deployment agent is not able to set the Framework properties it may fail.

5.1.4 Variables

Configurations and Framework Properties support late binding of values. This enables setting these items through the deployment agent~~at launch time~~, for example to specify a database user name or other information that may be variable between runtimes.

5.1.5 Prototype

If a Feature is similar to another Feature, it can use the other Feature as its prototype. Entities can be removed from the prototype and additional entities can be added in the resulting Feature. Prototypes can be used to create variants of existing Features. The newly created Features must have a different ID than the prototype they're based on.

~~It is possible to prevent a feature from being used as a prototype by marking it as *final*.~~

5.1.6 Extensibility

Features are extensible. This means that custom content can be provided inside the Feature Model. The custom content can be used to store extra metadata in the Feature, or to define additional processing associated with the Feature. Extensions can be handled through plugins or external tools at various times in the Feature lifecycle.

Extension content can be provided in the following types:

- TEXT: the content is provided as plain text.
- JSON: the content is specified as custom JSON.
- ARTIFACTS: the content declares a list of artifacts, similar to the bundles listed in the Feature.

Additionally, custom extensions can declare whether they need to be handled during processing. ~~by the runtime. If the extension content is optional for the runtime, or to be handled by external tools, the extension can be declared as *optional*.~~ IExtensions are optional by default, if the extension requires that it must be handled by a plugin at ~~runtime~~during processing it can declare itself as *mandatory*. In this case, if no plugin is present to handle the extension, the entity processing the Feature should fail~~the feature is prevented from launching~~.

5.1.7 Aggregation

Multiple Features can be combined into a larger Feature. This can be useful to assemble a runtime or application out of a number of smaller Features. ~~The aggregation process can also produce features that are complete, which can therefore be run without the need for any additional features.~~

Suggestion: support aggregate definition files so that global aggregation can be done and provide the opportunity to re-make decisions made for previous aggregations. Maybe in the feature? Maybe as a requirement?

During aggregation conflict resolution may be necessary and merge strategies are needed. For example when multiple Features declare the same bundle in different versions or when multiple Features declare the same configuration PID. Additionally the aggregation process provides an opportunity for tooling to validate Features and/or the aggregation result. For example to see if the resulting aggregate is complete, or other.

Plugins can be provided that act on the merging of extensions. They can implement custom merging strategies, or otherwise act on the extensions. The operational details of these extensions are outside of the scope of this specification.

5.2 Feature Descriptor

The Feature Model is commonly described using a Feature JSON file.

BJ: wants SymbolicName + Version being the ID, not Maven GAV.

5.2.1 Example Feature Model

```
{
    "#": "A key that starts with a hash is a comment",
    "id": "org.foo.bar:my.app:1.0",

    "title": "A title for the feature. (optional)",
    "description": "A description for the feature. (optional)",
    "vendor": "The feature vendor, for example 'Apache Software Foundation'. (optional)",
    "license": "The license of this feature file, for example 'ASL-2'. (optional)",
    "location": "The location might be the location of the feature file or any other means identifying
where the object is defined. (optional)",
```

```
    "# A complete feature has no external dependencies": "(optional)",
    "complete": true,

    "# A final feature cannot be used as a prototype for another feature": "(optional)",
    "final": false,

    "# variables": "used in configuration and framework properties are substituted at launch time.",
    "variables": {
        "cfgvar": "somedefault",
        "org.abc.xyz": "1.2.3"
    },

    "# A prototype is another feature that is used as a prototype for this one ":
    "# Bundles, configurations and framework properties can be removed from the ",
    "# prototype. Bundles with the same artifact ID defined in the feature override ":
    "# bundles with this artifact ID in the Prototype",
    "prototype":
        {
            "id": "org.foo:some-other-feature:1.2.3",
            "removals": {
                "#": "Configurations, bundles and framework properties from the prototype can be
removed.",
                "configurations": [],
                "bundles": [],
                "framework-properties": []
            }
        },

    "# Requirements over and above the requirements in the bundles.": "",
    "requirements": [
        {
            "namespace": "osgi.contract",
            "directives": {
                "filter": "(&(osgi.contract=JavaServlet)(version=3.1))"
            }
        }
    ],

    "# Capabilities over and above the capabilities provided by the bundles referenced ":
    "# by the feature.",
    "capabilities": [
        {
            "namespace": "osgi.implementation",
            "attributes": {
                "osgi.implementation": "osgi.http",
                "version:Version": "1.1"
            },
            "directives": {
                "uses":
"javax.servlet,javax.servlet.http,org.osgi.service.http.context,org.osgi.service.http.whiteboard"
            }
        },
        {
            "namespace": "osgi.service",
            "attributes": {
                "objectClass:List<String>": "org.osgi.service.http.runtime.HttpServiceRuntime"
            },
            "directives": {
                "uses": "org.osgi.service.http.runtime,org.osgi.service.http.runtime.dto"
            }
        }
    ],

    "# Framework properties to be provided to the running OSGi Framework":"",
    "framework-properties": {
        "foo": 1,
```

```
        "org.osgi.framework.storage": "${tempdir}",
        "org.apache.felix.scr.directory": "launchpad/scr"
    },

    "# The bundles that are part of the feature. Bundles are referenced using Maven ":
    "# coordinates and can have additional metadata associated with them. Bundles can ",
    "# specified as either a simple string (the Maven coordinates of the bundle) or ":
    "# as an object with 'id' and additional metadata.",
    "bundles": [
        {
            "id": "org.foo.bar:util-bundle:2.2.0",
            "hash": "4632463464363646436",

            "#": "This is the relative start order inside the feature",
            "start-order": 5
        },
        {
            "id": "org.foo.bar:application-bundle:2.0.0",
            "start-order": 10
        },
        {
            "id": "org.foo.bar:another-bundle:2.1.0",

            "#": "OSGi start level is also supported",
            "start-level": 20
        },
        "org.foo.bar:foo-xyz:1.2.3"
    ],

    "# The configurations are specified following the format defined by the OSGi Configurator ":
    "# specification: https://osgi.org/specification/osgi.cmpn/7.0.0/service.configurator.html ",
    "# Variables declared in the variables section can be used for late binding of variables, ":
    "# they can be specified with the Launcher, or the default from the variables section is used.",
    "# Factory configurations can be specified using the named factory syntax, which separates ":
    "# The factory PID and the name with a tilde '~'",
    "configurations": {
        "my.pid": {
            "foo": 5,
            "something-enabled": false,
            "bar": "${cfgvar}",

            "# The tempdir variable is not specified at the variables section.":
            "# It needs to be provided at launch, otherwise the launch will stop.",
            "tempdir": "${tempdir}",


            "number:Integer": 7
        },
        "my.factory.pid~name": {
           "a.value":"yeah"
        }
    },
    "sql-init|TEXT:true": {
        "# create some database tables for this feature",
        "CREATE TABLE FOO (...)",
        "CREATE TABLE BAR (...)"
    },
    "my-metadata|JSON:false": {
        "scm-location": "git@github.com:myorg/myproj.git"
    }
}
```

5.3 ~~Feature Launching~~

~~To create a running process or application from one or more Feature files, these must be provided to a launcher tool. The launcher will perform the following process:~~

1. ~~Aggregate all provided Features to a single effective Feature. This aggregation is also executed if only a single Feature is launched to guarantee that all custom extension merge handlers are triggered.~~
2. ~~Apply overridden variables as provided to the launcher.~~
3. ~~Launch an OSGi framework. If Framework Properties are provided they are applied here.~~
4. ~~Install all bundles listed in the effective Feature.~~
5. ~~Start all bundles in the appropriate order.~~
6. ~~As soon as the Configuration Admin service becomes available, configurations listed in the Feature are applied.~~

5.4 ~~Specified Extensions~~

~~This specification lists a number of extensions which are optionally implemented as part of a Feature model implementation.~~

5.4.1 ~~API Regions~~

~~API Regions implement a mechanism to segregate the exported APIs into separate regions which are available to bundles importing this API in the Feature model.~~

~~API define named regions that hold the API… TODO finish this section.~~

5.4.2 ~~…~~

~~… other specified extensions? …~~

6 Data Transfer Objects

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

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

The DTOs for the design in this RFC should be described here and if there are no DTOs being defined an explanation should be given explaining why this is not applicable in this case.

This section is optional and could also be provided in a separate RFC.

7 Javadoc

Please include Javadoc of any new APIs here, once the design has matured. Instructions on how to export Javadoc for inclusion in the RFC can be found here: https://www.osgi.org/members/RFC/Javadoc

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

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

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

Add references simply by adding new items. You can then cross-refer to them by chosing <Insert><Cross Reference><Numbered Item> and then selecting the paragraph. ***STATIC REFERENCES (I.E. BODGED) ARE NOT ACCEPTABLE, SOMEONE WILL HAVE TO UPDATE THEM LATER, SO DO IT PROPERLY NOW.***

10.2 Author's Address

Name	David Bosschaert
Company	Adobe
Address	
Voice	
e-mail	bosschae@adobe.com

Name	
Company	
Address	
Voice	
e-mail	

Name	
Company	
Address	
Voice	
e-mail	

10.3 Acronyms and Abbreviations

10.4 End of Document