RFC 222: Declarative Services Updates

Draft

12 Pages

Abstract

Updates to Declarative Services for Release 7.

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	2016-04-13	Initial draft BJ Hargrave, IBM
2nd draft	2016-04-14	Add design for component reclamation and field injection of component activation objects. BJ Hargrave IBM

1 Introduction

This RFC collects a numbers of requested enhancements to Declarative Services that were suggested after Release 6 design work was completed.

2 Application Domain

Declarative Services (DS) was first released in 2005 as part of Release 4. From the Version 1.0 spec:

> The service component model uses a declarative model for publishing, finding and binding to OSGi services. This model simplifies the task of authoring OSGi services by performing the work of registering the service and handling service dependencies. This minimizes the amount of code a program- mer has to write; it also allows service components to be loaded only when they are needed. As a result, bundles need not provide a BundleActivator class to collaborate with others through the service registry.

DS has proven a popular and useful way of developing for OSGi. There have been 3 updates to the spec resulting in the current Version 1.3 in Release 6.

3 Problem Description

3.1 Component Factory Properties (Bug 2800)

Currently factory components can only have 2 service properties, component.name and component.factory. See 112.2.4 Factory Component.

It would be useful to allow a ComponentFactory service to have additional service properties. For example, a discussion of possible Device Access changes resulted in an alternate proposal using ComponentFactory. But this proposal utilized some service properties on the ComponentFactory service. Currently this could only be done through the value of the factory attribute which results in the component.factory service property.

3.2 Component Reclamation (Bug 2801)

With the current DS spec, a service can either be lazy or immediate. Neglecting configuration policy and satisfying of references, an immediate service is activate as soon as possible and deactivated when the bundle is stopped. A lazy component is only activated if someone else is using it, and deactivated once it's not used anymore. For the examples below I used Event Admin, as everyone is familiar with it; but it's applicable for other scenarios, usually whiteboard related.

There are at least two consequences of the lazy behavior:

1. A lazy component might create a burden on the system. For example, if an EventHandler is lazy and the handler is activated and deactivated for each event it's receiving, a lot of activation/deactivation of that service might happen, even concurrently. Of course, an event admin implementation can keep the service once it's send the first event. Making the EventHandler immediate reduces the burden in any case.

2. If a service wants to store information in between usages, for example if an EventHandler wants to count how often it was invoked, immediate is the only option. Of course, if the service becomes unsatisfied or the bundle is restarted the state is lost. However, in many cases keeping state in this way is sufficient.

For use case like the above mentioned, immediate works but comes with the penalty that the service is activated as soon as possible, even if it is not used. For example, if there is no EventAdmin the EventHandler is activated nevertheless.

Therefore it would be nice to have an option in between immediate and lazy: the service is activated like it is lazy but deactivated like it is immediate.

3.3 Constructor Injection (Public Bug 179)

Method injection was the original dependency injection technique supported in DS. In Version 1.3, field injection support was added. Both of these techniques require the use of non-final field since the fields must be updatable after object construction. There is interest in also supporting constructor injection to allow the injected component instances to be stored in final fields.

3.4 Mapped Field Injection (RFP 178)

When having multiple instances of the same service interface, often a key property is used to identify and differentiate these instances. When using these service instances from a DS component, it makes sense to collect these services and map them by their key property.

For example using the good old method injection:

```
Map<String, SomeService> services = new ConcurrentHashMap<>();

@Reference(
      cardinality=ReferenceCardinality.MULTIPLE,
      policy=ReferencePolicy.DYNAMIC)
void addService(SomeService s, Map<String, Object> properties){
    String key = (String)properties.get("keyProperty");
    services.put(key, s);
}

void removeService(SomeService s, Map<String, Object> properties){
    String key = (String)properties.get("keyProperty");
    services.remove(key);
```

```
}
```

However, this is currently not possible using field injection.

3.5 Field injection of component activation objects (Bug 2902)

In many cases the activate method is only implemented to receive the ComponentContext, BundleContext or configuration and store it in a field. Similarly the deactivate method might be implemented to null out these fields - this allows service methods to check whether a component is active or not.

To reduce this boilerplate code, we could support annotating fields with @Activate. The type of a field can be one of the types supported by the activate method and are set before any component method is called.

4 Requirements

DS-0010 – Provide a means to define configurable services properties for ComponentFactory services. These are separate from the service properties of the component instances constructed by the ComponentFactory service.

DS-0020 – Provide a means to specify that a component instance construction can be delayed but its reclamation deferred until the component configuration becomes unsatisfied.

DS-0030 – Provide a means to support injecting bound services to a component constructor.

DS-0040 – Provide a means to inject a map of keyed services. The means must allow the key name to be specified and also whether the multiple values for the key are supported.

DS-0041 – The type of the key and the type of the value, for DS-0040, must be specifiable or inferred from the generics types of annotated Map field.

DS-0050 – Provide a means to inject component activation objects into fields. This means must also handle modified and deactivation cases.

DS-1000 – All solutions must provide a way to utilize them via Annotations as well as via the component description xml.

5 Technical Solution

5.1 Schema namespace update

The XML schema namespace is updated to `http://www.osgi.org/xmlns/scr/v1.4.0` for the new features being added below.

5.2 Component Factory Properties

TBD

5.3 Component Reclamation

A new `activation-policy` attribute is defined for the `<component>` element. This attribute defines the policy for activating and deactivating the component.The `activation-policy` attribute replaces the `immediate` attribute which is removed from the schema.

The `activation-policy` attribute can have one of the following values:

- `immediate` – The component instance must be activated as soon as the component configuration is satisfied. The component instance must remain activated until the component configuration becomes unsatisfied when the component instance must be deactivated. This is the replacement for `immediate=true`.

- `ondemand` – The activation of a component instance must be delayed until there is an actual need for the component instance such as an actual request for the service object. The component instance must remain activate as long as the component instance is in use. If the service registered by a component configuration becomes unused because there are no more bundles using it, then SCR should deactivate the component instance. This allows SCR implementations to eagerly reclaim activated component configurations. This is the replacement for `immediate=false`.

- `delayed` – The activation of a component instance must be delayed until there is an actual need for the component instance such as an actual request for the service object. The component instance must remain activated until the component configuration becomes unsatisfied when the component instance must be deactivated. This is a new policy.

The default policy is `immediate` if the component is not a factory component and does not specify a service. Otherwise the default policy is `ondemand`.

Both the `ondemand` and `delayed` policies delay activation of component instances until they are actually needed but the `delayed` policy will keep the component instance activated until it becomes unsatisfied while the `ondemand` policy will allow SCR to deactivate component instances which are not in use as services.

The `Component` annotation is updated to add a new `activationPolicy` element of type `ActivationPolicy` which is an enum having the values: `IMMEDIATE`, `ONDEMAND`, and `DELAYED`. The `immediate` element of the `Component` annotation is deprecated. If the `activationPolicy` element is specified, then the `immediate` element is ignored.

5.4 Constructor Injection

TBD

5.5 Mapped Field Injection

TBD

5.6 Field injection of component activation objects

A new `activation-fields` attribute is defined for the `<component>` element which names the instance fields in the component implementation class which are to be injected with component activation objects. This attribute must contain a whitespace separated list of field names.

An activation field must be one of the following types:

- `ComponentContext` - The field will be set to the Component Context for the component configuration.
- `BundleContext` - The field will be set the Bundle Context of the component's bundle.
- `Map` - The field will be set with an unmodifiable Map containing the component properties.
- A component property type - The field will be set with an instance of the component property type which allows type safe access to component properties defined by the component property type.

Only non-final instance fields of the field types above are supported. If an activation field is declared with the `static` modifier, the `final` modifier, or has a type other than one of the above, SCR must log an error message with the Log Service, if present, and the field must not be modified.

When using activation fields, SCR must modify the activation fields in the component instance at the appropriate time. SCR must modify the fields on the following component life cycle events:

- activation - The field is set to the component activation object. The field must be set after the component instance constructor completes and before any other method, such as the activate method, is called. That is, there is a *happens-before* relationship between the field being set and any method being called on the fully constructed component instance.
- modification - For fields of type `Map` and component property types which are declared with the `volatile` modifier, the field is set to the modified component properties before the modified method, if specified, is called. If the field is not declared with the `volatile` modifier, it is not modified. The field is only modified if declared with the `volatile` modifier so that field value changes made by SCR are visible to other threads.
  If the component does not specify a modified method or an activation field of type `Map` or a component property type which is declared `volatile`, then the component configuration will become unsatisfied if its component properties are modified since there is no way for SCR to provide the modified component properties to the component instance.
- deactivation - For fields which are declared with the `volatile` modifier, the field is set to `null` after the deactivate method, if specified, completes. If the field is not declared with the `volatile` modifier, it is not modified. The field is only modified if declared with the `volatile` modifier so that field value changes made by SCR are visible to other threads.

Care must be taken by the component implementation regarding the field. SCR has no way to know if the component implementation itself may alter the field value. The component implementation should not alter the field value and allow SCR to manage it. SCR must treat the field as if the component implementation does not alter the field value so SCR may retain its own copy of the value set in the field.

Fields can be declared private in the component class but are only looked up in the inheritance chain when they are protected, public, or have default access.

The `Activate` annotation is modified to allow it to be applied to fields. Applying the `Activate` annotation to a field will add that field to the `activation-field` attribute of the `<component>` element. Multiple fields can be annotated with `Activate` as well as an activate method.

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

None at this time.

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

10.2 Author's Address

Name	BJ Hargrave
Company	IBM

10.3 Acronyms and Abbreviations

DS – Declarative Services

SCR – Service Component Runtime

10.4 End of Document