RFC 222: Declarative Services Updates

Final

94 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://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 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	04/13/2016	Initial draft BJ Hargrave, IBM
2nd draft	04/14/2016	Add design for component reclamation and field injection of component activation objects. BJ Hargrave IBM

Revision	Date	Comments
3rd draft	04/19/2016	After review at CPEG meeting. Updated design for component reclamation and field injection of component activation objects. Added use case for constructor injections. Added Logger support from RFC 219. BJ Hargrave, IBM
4th draft	06/27/2016	Add design for Mapped Field Injection based on RFP-178. Carlos Sierra, Liferay Raymond Augé, Liferay
5th draft	5 Sep 2016	Added initial proposal for constructor injection design. Replaced the design for partitioned maps. BJ Hargrave, IBM
6th draft	19 Sep 2016	Comments from CPEG f2f meeting in San Jose. Added section for ConfigurationPlugin support changes, Converter package usage. Removed section on partitioned maps. Added updated to DTOs and Javadoc. BJ Hargrave, IBM
7th draft	19 Oct 2016	Add ComponentPropertyType annotation as suggested by Bug 2952 BJ Hargrave, IBM
8th draft	20 Oct 2016	Added a set of standard component property types for the user settable service properties in the framework Constants type. The example is updated to use some of those standard component properly types. BJ Hargrave, IBM
9th draft	08 Nov 2016	Added conversion rule for String→Class since the Converter package does not include that in the default conversions. Added design for factory properties. BJ Hargrave, IBM
10th draft	09 Nov 2016	Add failure information to dto as suggested by Bug 2953. BJ Hargrave, IBM
11th draft	21 Jun 2017	Moved Converter discussion to considered alternatives. BJ Hargrave, IBM
Final draft	2018-01-08	Final for RFC voting 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 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 (Bug 2790, 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.

Also, a component implementation super type constructor may require objects such as component activation objects or bound services or information obtainable from them. Supporting constructor injection of component activation objects and bound services will support this.

3.4 Mapped Field Injection (Bug 2940)

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);
}
```

Field injection using a custom list or collection of Map.Entry can be used, but the Reference annotation does not allow specifying the field-collection-type meaning xml authoring of the component description is needed.

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.

3.6 Logger Support

RFC 219 Log Service Update adds support for named loggers. Since logging is both important and needed early in code execution, DS must add special support for injecting Logger and FormatterLogger objects even though they themselves are not services.

3.7 Specifying component properties via annotations (Bug 2952)

The DS 1.3 spec introduced the concept of Component Property Types as a type safe way to access component properties as well as to specify default values for component properties. Component Property Types could be used as parameter types in component lifecycle methods. At runtime, SCR must pass objects of those types as arguments whose methods return values based upon the component properties. During tool time, the component property types used as parameter types in the lifecycle methods are processed so that any default values specified in the annotation declaration are mapped into property elements in the generated component description XML.

However, there is no way to use the component property types to specific configuration property values other than the default values. This is not useful when multiple component implementation may which to use the same component property types but specify different component property values. In this case, the Component annotation would have to specify property values in String form which loses the type safety of the component property types.

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

DS-0031 – Provide a means to support injecting component activation objects to a component constructor.

DS-0040 – Provide a means to support a map of keyed services.

DS-0050 – Provide a means to inject component activation objects into fields.

DS-0060 – Provide a means to inject Logger objects into a service component where the Logger objects are obtained from the LoggerFactory by SCR.

DS-0070 – Provide a means to use component property types to specify component property values for a component which are different then the default values.

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

5 Technical Solution

5.1 Version Increases

The XML schema namespace is updated to `http://www.osgi.org/xmlns/scr/v1.4.0` for the new features being added below. The package versions are updated to 1.4 also.

A constant is added to ComponentConstants to act as a compile time constant holding the specification version, 1.4, for use in annotations such as Version and Requirement.

Utilizing the new Requirement annotation from RFC 220, the Component annotation is annotated with

```
@Requirement(namespace = ExtenderNamespace.EXTENDER_NAMESPACE,
      name = ComponentConstants.COMPONENT_CAPABILITY_NAME,
      version = ComponentConstants.COMPONENT_SPECIFICATION_VERSION)
```

At tool time, a bundle using the Component annotation will have a Require-Capability for SCR added to its manifest.

5.2 Factory Properties

To allow the developer to specify service properties for the `ComponentFactory` service, two new elements are added to the component description schema as child elements of the `component` element:

```
<choice minOccurs="0" maxOccurs="unbounded">
    <element name="factoryProperty" type="scr:Tproperty" />
    <element name="factoryProperties" type="scr:Tproperties" />
</choice>
```

These elements can be used, when the `factory` attribute of the `component` element is specified. If the `factory` attribute of the `component` element is not specified, the component is not a factory component and the `factoryProperty` and `factoryProperties` elements are ignored if specified. If the `factory` attribute of the `component` element is specified, the `factoryProperty` and `factoryProperties` elements supply factory properties which are used as service properties for the `ComponentFactory` service in addition to the existing `component.name` and `component.factory` factory properties which are always set and cannot be overridden.

Factory properties are separate from component properties and are only specified in the component description. Factory properties are not component properties and are not passed to the component instance and are only used as service properties for the `ComponentFactory` service.

To enable factory properties to be specified via annotations, two new elements are added to the `Component` annotation:

```
String[] factoryProperty() default {};
String[] factoryProperties() default {};
```

If either of these elements is specified, then the `factory` element must also be specified to mark the component as a factory component. The values of these elements have the same syntax as the values of the `property` and `properties` elements, respectively.

5.3 Component Reclamation

Prototype scope service component instances must be reclaimed when released since they cannot be used again. Singleton scope service component instances may be reused by any bundle after being released and bundle scope service component instances may be reused by the same bundle again after being released.

Section 112.5.4 Delayed Component is updated to replace:

> If the service registered by a component configuration becomes unused because there are no more bundles using it, then SCR should deactivate that component configuration. This allows SCR implementations to eagerly reclaim activated component configurations.

with

> If the service has the `scope` attribute set to `prototype`, SCR must deactivate a component configuration when it stops being used as a service object since the component configuration must not be reused as a service object. If the service has the `scope` attribute set to `singleton` or `bundle`, SCR must deactivate a component configuration when it stops being used as a service object after a delay since the component configuration may be reused as a service object in the near future. This allows SCR implementations to reclaim component configurations not in use while attempting to avoid deactivating a component configuration only to have to quickly activate a new component configuration for a new service request. The delay amount is implementation specific and may be zero.

5.4 Constructor Injection

The spec is updated to allow a constructor as an activate method. By specifying the activate method name as “-init-”, SCR must use a constructor declared on the implementation class as the activate method.

The spec is further updated to allow activate method parameters to be referenced services. Thus referenced services can be injected into the activate method, and since the activate method can now be a constructor, referenced services can be injected into a constructor. <reference> elements will use the new “parameter” attribute to denote the reference is to an activate method parameter. The value of the attribute is the zero-based position of the parameter in the method declaration. Activate method parameters which do not have a corresponding <reference> element are normal activate method arguments. The rules for locating the activate method will be amended to include <reference> elements using the parameter attribute.

<reference> elements with the parameter attribute must have policy=STATIC since the activate method is only called once per component instance activation.

During component activation, the following steps are taken:

1. Load the component implementation class.
2. Create the component context.
3. If there is no activate method or the activate method is not a constructor, call the default constructor to create the component instance. Otherwise, if there is an activate method and the activate method is a constructor, bind the target services referenced by the activate constructor and call the activate constructor to create the component instance.
4. Bind the target services not reference by the activate constructor, if any.

5. If there is an activate method and the activate method is not a constructor, call the activate method.

For constructor parameters with optional and unary references, if there is no bound service, then null will be passed as the parameter value.

See the following example code which uses annotations to declare a constructor as the activate method with references services as arguments.

```
@Component
public class ConstructorInjection {
	@interface Config {
		int port() default 80;
	}

	private final LogService		log;
	private final Config			config;
	private final ComponentContext	cc;
	private final List<EventListener>	listeners;

	// Update @Activate so it can be applied to a CONSTRUCTOR
	// Only one method or constructor can be marked @Activate
	// Allow activation methods to have @Reference annotated parameters
too
	/*
	 * <scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.4.0"
	 * name="testConstructorInjection" activate="-init-"
	 * deactivate="deactivate">
	 */
	@Activate
	public ConstructorInjection( //
			// Update @Reference so it can be applied to a PARAMETER;
			// policy=STATIC
			/*
			 * <reference name="log"
			 * interface="org.osgi.service.log.LogService"
			 * parameter=0/>
			 */
			@Reference LogService log, //

			// Non-@Reference annotated parameters are activate method
args
			Config config, //
			ComponentContext cc, //
```

```
                /*
                 * <reference name="listeners" cardinality="0..n"
                 * interface="java.util.EventListener" parameter=3
                 * field-collection-type="service"/>
                 */
                @Reference List<EventListener> listeners //
        ) {
                this.log = log;
                this.config = config;
                this.cc = cc;
                this.listeners = listeners;
                System.out.println("Hello World!");
        }

        /**
         */
        @Deactivate
        private void deactivate() {
                System.out.println("Goodbye World!");
        }
}
```

5.5 Reference annotation support for specifying field-collection-type

Using a custom implementation of list or collection of Map.Entry with field-option=update can be used to support partitioning multiple services based upon any criteria. The add and remove methods of the custom implementation will be called with the service properties and the service object for each service added or removed from the custom implementation. The implementation can then partition the services in any desired way such as by the value of a specific service property.

A change to the Reference annotation is needed to allow the programmer to specify the field-collection-type rather than it being only inferred by the tool processing the annotation. A new collectionType element is added to the Reference annotation to allow the field-collection-type to be specified.

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 set the activation fields in the component instance at component activation. The fields 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 fields being set and any method being called on the fully constructed component instance.

A modified method must be specified if the component requires notification of component property modification. A deactivate method must be specified if the component requires notification of deactivation.

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.

5.7 Logger Support

DS must add special support for injecting `Logger` and `FormatterLogger` objects even though they themselves are not services. When a component references the `Logger` or `FormatterLogger` types, SCR must get first get the `LoggerFactory` service matching the reference and then call the `getLogger(Bundle, String, Class)` method passing the bundle declaring the component as the first argument, the component implementation class name as the second argument and the `Logger` type as the third argument. The returned `Logger` object is then injected for the reference, rather than the `LoggerFactory` service used to create the Logger.

A DS example using Logger:

```
@Component
public class MyComponent {
  @Reference
  private Logger logger;
  @Activate
  void activate(ComponentContext context) {
    logger.trace("activating component id {}",
      context.getProperties().get("component.id"));
  }
}
```

5.8 Improved ConfigurationPlugin Support

RFC 227 include several enhancements to Configuration Admin. Of particular note for DS is the improvements to ConfigurationPlugin support which will allow SCR to get Configurations after ConfigurationPlugins have been able to mutate the configuration data.

The DS spec will be updated to require SCR to obtain the configuration data from the new Configuration.getProcessedProperties method. SCR must already use the component's BundleContext to obtain the configuration. The use the new getProcessedProperties method, SCR must supply a ServiceReference for a ManagedService or ManagedServiceFactory. So SCR must register the ManagedService or ManageServiceFactory service using the component's BundleContext so the ServiceReference for that service can be used as the argument for the getProcessedProperties method. SCR should register these services without a service.pid service property if the SCR implementation will obtain the Configuration object through other means such as a method on the ConfigurationAdmin service so that the service is not called by ConfigurationAdmin.

5.9

5.10 Using Component Property Types as annotations on Components

A new ComponentPropertyType meta-annotation is defined which can be applied to component property types. This meta-annotation will then allow component property types to be applied to the implementation class of the component (along side the Component annotation) to declare component property values. When the Component annotation is processed at tool time, any annotations also applied to the implementation class of the component which are meta-annotated with ComponentPropertyType are processed for component property values using either the value specified in the annotation use or the default value.

If an element of a component property type specifies a value of an empty array, then the element is not used to generate a property value in generated component description. If the type of an an element of a component property type is an annotation or array of annotations, then the element is not used to generate a property value in generated component description.

Section 112.8.2 will be updated to specify this additional method to specifying component property values.

Section 112.8.3 will be updated to add a step between the current steps 1 and 2 for the use of component property types as annotations on the implementation class of the component. These annotations must processed in the order they appear in the class file. However, the order of the RuntimeVisibleAnnotations and RuntimeInvisibleAnnotations attributes in the class file is unspecified.

Also, a new org.osgi.service.component.propertytypes package is added which defines a set of component property types for user settable service properties from the framework Constants type. This includes ServiceRanking, ServiceDescription, ServiceVendor and ExportedService. The latter includes the set of service properties which can be used when marking a service for export by a remote services implementation.

In the following example, the ServiceRanking and ServiceDescription component property types are used to annotate the implementation class of the component and the user defined Config component property type is used as a parameter type of a life cycle method.

```
@interface Config {
      boolean enabled() default true;
      String[] names() default {"a", "b"};
      String topic() default "default/topic";
}
@Component
@ServiceRanking(service_ranking=100)
@ServiceDescription(service_description=”My Acme Service implementation”)
public class MyComponent implements AcmeService {
      @Activate
      void activate(Config config) {
            if (config.enabled()) {
                  // do something
            }
            for (String name : config.names()) {
                  // do something with each name
```

```
                }
            }
        }
```

When process at tool time, the generated component description XML will contain property declarations like:

```
<property name=”service.ranking” type=”Integer” value=”100”/>
<property name=”service.description” value=”My Acme Service implementation”/>
<property name=”enabled” type=”Boolean” value=”true”/>
<property name=”names”>a
b</property>
<property name=”topic” value=”default/topic”/>
```

6 Data Transfer Objects

The ReferenceDTO is updated to add 2 new fields: parameter and collectionType for the new parameter attribute and the existing field-collection-type attribute which can now be explicitly set via the Reference annotation.

The ComponentDescriptionDTO is updated to add 2 new fields: factoryProperties and activationFields. If the component is a factory component, the factoryProperties field must be non-null and contain any factory properties declared in the component description. If the component declares activation fields, the activationFields field must contain the field names.

The ComponentConfigurationDTO is updated to add 2 new fields: failure and service. A FAILED_ACTIVATION state value is also added. If the component configuration is satisfied but fails to activate due to some exception, the state must be FAILED_ACTIVATION and the failure field will hold the exception information. The service field holds a ServiceReferenceDTO to the component’s service, if any.

7 Javadoc

OSGi Javadoc

1/8/18 1:50 PM

Package Summary		Page
org.osgi.service.component	Service Component Package Version 1.4.	17
org.osgi.service.component.annotations	Service Component Annotations Package Version 1.4.	32
org.osgi.service.component.propertytypes	Component Property Types Package Version 1.4.	66
org.osgi.service.component.runtime	Service Component Runtime Package Version 1.4.	72
org.osgi.service.component.runtime.dto	Service Component Runtime Data Transfer Objects Package Version 1.4.	76

Package org.osgi.service.component

```
@org.osgi.annotation.versioning.Version(value="1.4.0")
```

Service Component Package Version 1.4.

See:

Description

Interface Summary		Page
ComponentConstants	Defines standard names for Service Component constants.	18
ComponentContext	A Component Context object is used by a component instance to interact with its execution context including locating services by reference name.	22
ComponentFactory	When a component is declared with the `factory` attribute on its `component` element, Service Component Runtime will register a Component Factory service to allow new component configurations to be created and activated rather than automatically creating and activating component configuration as necessary.	28
ComponentInstance	A ComponentInstance encapsulates a component instance of an activated component configuration.	29
ComponentServiceObjects	Allows multiple service objects for a service to be obtained.	30

Exception Summary		Page
ComponentException	Unchecked exception which may be thrown by Service Component Runtime.	26

Package org.osgi.service.component Description

Service Component Package Version 1.4.

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.component; version="[1.4,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.component; version="[1.4,1.5)"
```

Interface ComponentConstants

org.osgi.service.component

```
@org.osgi.annotation.versioning.ProviderType
public interface ComponentConstants
```

Defines standard names for Service Component constants.

Field Summary		Page
String	**COMPONENT_CAPABILITY_NAME** Capability name for Service Component Runtime.	21
String	**COMPONENT_FACTORY** A service registration property for a Component Factory that contains the value of the `factory` attribute.	19
String	**COMPONENT_ID** A component property that contains the generated id for a component configuration.	19
String	**COMPONENT_NAME** A component property for a component configuration that contains the name of the component as specified in the `name` attribute of the `component` element.	19
String	**COMPONENT_SPECIFICATION_VERSION** Compile time constant for the Specification Version of Declarative Services.	21
int	**DEACTIVATION_REASON_BUNDLE_STOPPED** The component configuration was deactivated because the bundle was stopped.	21
int	**DEACTIVATION_REASON_CONFIGURATION_DELETED** The component configuration was deactivated because its configuration was deleted.	21
int	**DEACTIVATION_REASON_CONFIGURATION_MODIFIED** The component configuration was deactivated because its configuration was changed.	20
int	**DEACTIVATION_REASON_DISABLED** The component configuration was deactivated because the component was disabled.	19
int	**DEACTIVATION_REASON_DISPOSED** The component configuration was deactivated because the component was disposed.	21
int	**DEACTIVATION_REASON_REFERENCE** The component configuration was deactivated because a reference became unsatisfied.	20
int	**DEACTIVATION_REASON_UNSPECIFIED** The reason the component configuration was deactivated is unspecified.	19
String	**REFERENCE_TARGET_SUFFIX** The suffix for reference target properties.	19
String	**SERVICE_COMPONENT** Manifest header specifying the XML documents within a bundle that contain the bundle's Service Component descriptions.	18

Field Detail

SERVICE_COMPONENT

```
public static final String SERVICE_COMPONENT = "Service-Component"
```

Manifest header specifying the XML documents within a bundle that contain the bundle's Service Component descriptions.

The attribute value may be retrieved from the `Dictionary` object returned by the `Bundle.getHeaders` method.

COMPONENT_NAME
```
public static final String COMPONENT_NAME = "component.name"
```

A component property for a component configuration that contains the name of the component as specified in the `name` attribute of the `component` element. The value of this property must be of type `String`.

COMPONENT_ID
```
public static final String COMPONENT_ID = "component.id"
```

A component property that contains the generated id for a component configuration. The value of this property must be of type `Long`.

The value of this property is assigned by Service Component Runtime when a component configuration is created. Service Component Runtime assigns a unique value that is larger than all previously assigned values since Service Component Runtime was started. These values are NOT persistent across restarts of Service Component Runtime.

COMPONENT_FACTORY
```
public static final String COMPONENT_FACTORY = "component.factory"
```

A service registration property for a Component Factory that contains the value of the `factory` attribute. The value of this property must be of type `String`.

REFERENCE_TARGET_SUFFIX
```
public static final String REFERENCE_TARGET_SUFFIX = ".target"
```

The suffix for reference target properties. These properties contain the filter to select the target services for a reference. The value of this property must be of type `String`.

DEACTIVATION_REASON_UNSPECIFIED
```
public static final int DEACTIVATION_REASON_UNSPECIFIED = 0
```

The reason the component configuration was deactivated is unspecified.

Since:
1.1

DEACTIVATION_REASON_DISABLED
```
public static final int DEACTIVATION_REASON_DISABLED = 1
```

The component configuration was deactivated because the component was disabled.

Since:
1.1

DEACTIVATION_REASON_REFERENCE
```
public static final int DEACTIVATION_REASON_REFERENCE = 2
```

The component configuration was deactivated because a reference became unsatisfied.

Since:
1.1

DEACTIVATION_REASON_CONFIGURATION_MODIFIED
```
public static final int DEACTIVATION_REASON_CONFIGURATION_MODIFIED = 3
```

The component configuration was deactivated because its configuration was changed.

Since:
1.1

DEACTIVATION_REASON_CONFIGURATION_DELETED
`public static final int` **`DEACTIVATION_REASON_CONFIGURATION_DELETED`** `= 4`

The component configuration was deactivated because its configuration was deleted.

Since:
1.1

DEACTIVATION_REASON_DISPOSED
`public static final int` **`DEACTIVATION_REASON_DISPOSED`** `= 5`

The component configuration was deactivated because the component was disposed.

Since:
1.1

DEACTIVATION_REASON_BUNDLE_STOPPED
`public static final int` **`DEACTIVATION_REASON_BUNDLE_STOPPED`** `= 6`

The component configuration was deactivated because the bundle was stopped.

Since:
1.1

COMPONENT_CAPABILITY_NAME
`public static final String` **`COMPONENT_CAPABILITY_NAME`** `= "osgi.component"`

Capability name for Service Component Runtime.

Used in `Provide-Capability` and `Require-Capability` manifest headers with the `osgi.extender` namespace. For example:

```
Require-Capability: osgi.extender;
 filter:="(&(osgi.extender=osgi.component)(version>=1.4)(!(version>=2.0)))"
```

Since:
1.3

COMPONENT_SPECIFICATION_VERSION
`public static final String` **`COMPONENT_SPECIFICATION_VERSION`** `= "1.4.0"`

Compile time constant for the Specification Version of Declarative Services.

Used in `Version` and `Requirement` annotations. The value of this compile time constant will change when the specification version of Declarative Services is updated.

Since:
1.4

Interface ComponentContext

org.osgi.service.component

```
@org.osgi.annotation.versioning.ProviderType
public interface ComponentContext
```

A Component Context object is used by a component instance to interact with its execution context including locating services by reference name. Each component instance has a unique Component Context.

A component instance may obtain its Component Context object through its activate, modified, and deactivate methods.

ThreadSafe

Method Summary		Page
void	**disableComponent**(String name) Disables the specified component name.	24
void	**enableComponent**(String name) Enables the specified component name.	24
org.osgi.framework.BundleContext	**getBundleContext**() Returns the BundleContext of the bundle which declares this component.	24
ComponentInstance<S>	**getComponentInstance**() Returns the Component Instance object for the component instance associated with this Component Context.	24
Dictionary<String,Object>	**getProperties**() Returns the component properties for this Component Context.	22
org.osgi.framework.ServiceReference<?>	**getServiceReference**() If the component instance is registered as a service using the service element, then this method returns the service reference of the service provided by this component instance.	25
org.osgi.framework.Bundle	**getUsingBundle**() If the component instance is registered as a service using the servicescope="bundle" or servicescope="prototype" attribute, then this method returns the bundle using the service provided by the component instance.	24
S	**locateService**(String name) Returns the service object for the specified reference name.	23
S	**locateService**(String name, org.osgi.framework.ServiceReference<S> reference) Returns the service object for the specified reference name and ServiceReference.	23
Object[]	**locateServices**(String name) Returns the service objects for the specified reference name.	23

Method Detail

getProperties

```
Dictionary<String,Object> getProperties()
```

Returns the component properties for this Component Context.

Returns:

The properties for this Component Context. The Dictionary is read only and cannot be modified.

---

locateService

```
S locateService(String name)
```

Returns the service object for the specified reference name.

If the cardinality of the reference is `0..n` or `1..n` and multiple services are bound to the reference, the service with the highest ranking (as specified in its `Constants.SERVICE_RANKING` property) is returned. If there is a tie in ranking, the service with the lowest service id (as specified in its `Constants.SERVICE_ID` property); that is, the service that was registered first is returned.

Parameters:

`name` - The name of a reference as specified in a `reference` element in this component's description.

Returns:

A service object for the referenced service or `null` if the reference cardinality is `0..1` or `0..n` and no bound service is available.

Throws:

`ComponentException` - If Service Component Runtime catches an exception while activating the bound service.

---

locateService

```
S locateService(String name,
                org.osgi.framework.ServiceReference<S> reference)
```

Returns the service object for the specified reference name and `ServiceReference`.

Type Parameters:

`S` - Type of Service.

Parameters:

`name` - The name of a reference as specified in a `reference` element in this component's description.

`reference` - The `ServiceReference` to a bound service. This must be a `ServiceReference` provided to the component via the bind or unbind method for the specified reference name.

Returns:

A service object for the referenced service or `null` if the specified `ServiceReference` is not a bound service for the specified reference name.

Throws:

`ComponentException` - If Service Component Runtime catches an exception while activating the bound service.

---

locateServices

```
Object[] locateServices(String name)
```

Returns the service objects for the specified reference name.

Parameters:

`name` - The name of a reference as specified in a `reference` element in this component's description.

Returns:

An array of service objects for the referenced service or `null` if the reference cardinality is `0..1` or `0..n` and no bound service is available. If the reference cardinality is `0..1` or `1..1` and a bound service is available, the array will have exactly one element.

Throws:

`ComponentException` - If Service Component Runtime catches an exception while activating a bound service.

---

getBundleContext

`org.osgi.framework.BundleContext` **`getBundleContext`**`()`

Returns the `BundleContext` of the bundle which declares this component.

Returns:

The `BundleContext` of the bundle declares this component.

---

getUsingBundle

`org.osgi.framework.Bundle` **`getUsingBundle`**`()`

If the component instance is registered as a service using the `servicescope="bundle"` or `servicescope="prototype"` attribute, then this method returns the bundle using the service provided by the component instance.

This method will return `null` if:

- The component instance is not a service, then no bundle can be using it as a service.
- The component instance is a service but did not specify the `servicescope="bundle"` or `servicescope="prototype"` attribute, then all bundles using the service provided by the component instance will share the same component instance.
- The service provided by the component instance is not currently being used by any bundle.

Returns:

The bundle using the component instance as a service or `null`.

---

getComponentInstance

`ComponentInstance<S>` **`getComponentInstance`**`()`

Returns the Component Instance object for the component instance associated with this Component Context.

Returns:

The Component Instance object for the component instance.

---

enableComponent

`void` **`enableComponent`**`(String name)`

Enables the specified component name. The specified component name must be in the same bundle as this component.

This method must return after changing the enabled state of the specified component name. Any actions that result from this, such as activating or deactivating a component configuration, must occur asynchronously to this method call.

Parameters:

`name` - The name of a component or `null` to indicate all components in the bundle.

---

disableComponent

`void` **`disableComponent`**`(String name)`

Disables the specified component name. The specified component name must be in the same bundle as this component.

This method must return after changing the enabled state of the specified component name. Any actions that result from this, such as activating or deactivating a component configuration, must occur asynchronously to this method call.

Parameters:

`name` - The name of a component.

---

getServiceReference

`org.osgi.framework.ServiceReference<?>` **`getServiceReference`**`()`

If the component instance is registered as a service using the `service` element, then this method returns the service reference of the service provided by this component instance.

This method will return `null` if the component instance is not registered as a service.

Returns:

The `ServiceReference` object for the component instance or `null` if the component instance is not registered as a service.

Class ComponentException

org.osgi.service.component

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └java.lang.RuntimeException
              └org.osgi.service.component.ComponentException
```

All Implemented Interfaces:

Serializable

```
public class ComponentException
extends RuntimeException
```

Unchecked exception which may be thrown by Service Component Runtime.

Constructor Summary	Page
`ComponentException(String message)` Construct a new ComponentException with the specified message.	26
`ComponentException(String message, Throwable cause)` Construct a new ComponentException with the specified message and cause.	26
`ComponentException(Throwable cause)` Construct a new ComponentException with the specified cause.	27

Method Summary		Page
`Throwable`	`getCause()` Returns the cause of this exception or `null` if no cause was set.	27
`Throwable`	`initCause(Throwable cause)` Initializes the cause of this exception to the specified value.	27

Constructor Detail

ComponentException

```
public ComponentException(String message,
                          Throwable cause)
```

Construct a new ComponentException with the specified message and cause.

Parameters:

`message` - The message for the exception.
`cause` - The cause of the exception. May be `null`.

ComponentException

```
public ComponentException(String message)
```

Construct a new ComponentException with the specified message.

Parameters:
`message` - The message for the exception.

ComponentException
`public` **`ComponentException`**`(Throwable cause)`

Construct a new ComponentException with the specified cause.

Parameters:
`cause` - The cause of the exception. May be `null`.

Method Detail

getCause
`public Throwable` **`getCause`**`()`

Returns the cause of this exception or `null` if no cause was set.

Overrides:
`getCause` in class `Throwable`
Returns:
The cause of this exception or `null` if no cause was set.

---

initCause
`public Throwable` **`initCause`**`(Throwable cause)`

Initializes the cause of this exception to the specified value.

Overrides:
`initCause` in class `Throwable`
Parameters:
`cause` - The cause of this exception.
Returns:
This exception.
Throws:
`IllegalArgumentException` - If the specified cause is this exception.
`IllegalStateException` - If the cause of this exception has already been set.

Interface ComponentFactory

org.osgi.service.component

Type Parameters:

`S` - Type of Service

```
@org.osgi.annotation.versioning.ProviderType
public interface ComponentFactory
```

When a component is declared with the `factory` attribute on its `component` element, Service Component Runtime will register a Component Factory service to allow new component configurations to be created and activated rather than automatically creating and activating component configuration as necessary.

ThreadSafe

Method Summary		Page
ComponentInstance<S>	**newInstance**(Dictionary<String,?> properties) Create and activate a new component configuration.	28

Method Detail

newInstance

ComponentInstance<S> **newInstance**(Dictionary<String,?> properties)

Create and activate a new component configuration. Additional properties may be provided for the component configuration.

Parameters:

`properties` - Additional properties for the component configuration or `null` if there are no additional properties.

Returns:

A `ComponentInstance` object encapsulating the component instance of the component configuration. The component configuration has been activated and, if the component specifies a `service` element, the component instance has been registered as a service.

Throws:

`ComponentException` - If Service Component Runtime is unable to activate the component configuration.

Interface ComponentInstance

org.osgi.service.component

Type Parameters:

S - Type of Service

```
@org.osgi.annotation.versioning.ProviderType
public interface ComponentInstance
```

A ComponentInstance encapsulates a component instance of an activated component configuration. ComponentInstances are created whenever a component configuration is activated.

ComponentInstances are never reused. A new ComponentInstance object will be created when the component configuration is activated again.

ThreadSafe

Method Summary		Page
void	dispose() Dispose of the component configuration for this component instance.	29
S	getInstance() Returns the component instance of the activated component configuration.	29

Method Detail

dispose

```
void dispose()
```

Dispose of the component configuration for this component instance. The component configuration will be deactivated. If the component configuration has already been deactivated, this method does nothing.

---

getInstance

```
S getInstance()
```

Returns the component instance of the activated component configuration.

Returns:

The component instance or `null` if the component configuration has been deactivated.

Interface ComponentServiceObjects

org.osgi.service.component

Type Parameters:
S - Type of Service

```
@org.osgi.annotation.versioning.ProviderType
public interface ComponentServiceObjects
```

Allows multiple service objects for a service to be obtained.

A component instance can receive a `ComponentServiceObjects` object via a reference that is typed `ComponentServiceObjects`.

For services with `prototype` scope, multiple service objects for the service can be obtained. For services with `singleton` or `bundle` scope, only one, use-counted service object is available.

Any unreleased service objects obtained from this `ComponentServiceObjects` object are automatically released by Service Component Runtime when the service becomes unbound.

Since:
1.3
See Also:
`org.osgi.framework.ServiceObjects`
ThreadSafe

Method Summary		Page
S	**getService**() Returns a service object for the associated service.	30
org.osgi.framework.ServiceReference<S>	**getServiceReference**() Returns the `org.osgi.framework.ServiceReference` for the service associated with this `ComponentServiceObjects` object.	31
void	**ungetService**(S service) Releases a service object for the associated service.	31

Method Detail

getService

```
S getService()
```

Returns a service object for the associated service.

This method will always return `null` when the associated service has been become unbound.

Returns:
A service object for the associated service or `null` if the service is unbound, the customized service object returned by a `ServiceFactory` does not implement the classes under which it was registered or the `ServiceFactory` threw an exception.
Throws:
`IllegalStateException` - If the component instance that received this `ComponentServiceObjects` object has been deactivated.
See Also:
`ungetService(Object)`

---

ungetService

`void` **`ungetService`**`(`S `service)`

Releases a service object for the `associated` service.

The specified service object must no longer be used and all references to it should be destroyed after calling this method.

Parameters:

`service` - A service object previously provided by this `ComponentServiceObjects` object.

Throws:

`IllegalStateException` - If the component instance that received this `ComponentServiceObjects` object has been deactivated.

`IllegalArgumentException` - If the specified service object was not provided by this `ComponentServiceObjects` object.

See Also:

`getService()`

---

getServiceReference

`org.osgi.framework.ServiceReference<`S`>` **`getServiceReference`**`()`

Returns the `org.osgi.framework.ServiceReference` for the service associated with this `ComponentServiceObjects` object.

Returns:

The `org.osgi.framework.ServiceReference` for the service associated with this `ComponentServiceObjects` object.

Package org.osgi.service.component.annotations

```
@org.osgi.annotation.versioning.Version(value="1.4.0")
```

Service Component Annotations Package Version 1.4.

See:

Description

Enum Summary		Page
CollectionType	Collection types for the `Reference` annotation.	34
ConfigurationPolicy	Configuration Policy for the `Component` annotation.	43
FieldOption	Field options for the `Reference` annotation.	46
ReferenceCardinality	Cardinality for the `Reference` annotation.	56
ReferencePolicy	Policy for the `Reference` annotation.	58
ReferencePolicyOption	Policy option for the `Reference` annotation.	60
ReferenceScope	Reference scope for the `Reference` annotation.	62
ServiceScope	Service scope for the `Component` annotation.	64

Annotation Types Summary		Page
Activate	Identify the annotated member as part of the activation of a Service Component.	33
Component	Identify the annotated class as a Service Component.	36
ComponentPropertyType	Identify the annotated annotation as a Component Property Type.	42
Deactivate	Identify the annotated method as the `deactivate` method of a Service Component.	45
Modified	Identify the annotated method as the `modified` method of a Service Component.	48
Reference	Identify the annotated member or parameter as a reference of a Service Component.	49

Package org.osgi.service.component.annotations Description

Service Component Annotations Package Version 1.4.

This package is not used at runtime. Annotated classes are processed by tools to generate Component Descriptions which are used at runtime.

Annotation Type Activate

org.osgi.service.component.annotations

```
@Retention(value=RetentionPolicy.CLASS)
@Target(value={
  ElementType.METHOD,
  ElementType.FIELD,
  ElementType.CONSTRUCTOR
})
public @interface Activate
```

Identify the annotated member as part of the activation of a Service Component.

When this annotation is applied to a:

- Method - The method is the `activate` method of the Component.
- Constructor - The constructor will be used to construct the Component and can be called with activation objects and bound services as parameters.
- Field - The field will contain an activation object of the Component. The field must be set after the constructor is called and before calling any other method on the fully constructed component instance. That is, there is a *happens-before* relationship between the field being set and calling any method on the fully constructed component instance such as the `activate` method.

This annotation is not processed at runtime by Service Component Runtime. It must be processed by tools and used to add a Component Description to the bundle.

Since:

1.1

See Also:

"The init, activate, and activation-fields attributes of the component element of a Component Description."

Enum CollectionType

org.osgi.service.component.annotations

```
java.lang.Object
  └ java.lang.Enum<CollectionType>
      └ org.osgi.service.component.annotations.CollectionType
```

All Implemented Interfaces:

Comparable<CollectionType>, Serializable

```
public enum CollectionType
extends Enum<CollectionType>
```

Collection types for the Reference annotation.

Since:

1.4

Enum Constant Summary	Page
PROPERTIES The properties collection type is used to indicate the collection holds unmodifiable Maps containing the service properties of the bound services.	35
REFERENCE The reference collection type is used to indicate the collection holds Service References for the bound services.	35
SERVICE The service collection type is used to indicate the collection holds the bound service objects.	34
SERVICEOBJECTS The serviceobjects collection type is used to indicate the collection holds Component Service Objects for the bound services.	35
TUPLE The tuple collection type is used to indicate the collection holds unmodifiable Map.Entries whose key is an unmodifiable Map containing the service properties of the bound service, as specified in PROPERTIES, and whose value is the bound service object.	35

Method Summary		Page
String	**toString**()	35
static CollectionType	**valueOf**(String name)	35
static CollectionType[]	**values**()	35

Enum Constant Detail

SERVICE

```
public static final CollectionType SERVICE
```

The service collection type is used to indicate the collection holds the bound service objects.

This is the default collection type.

REFERENCE

```
public static final CollectionType REFERENCE
```

The reference collection type is used to indicate the collection holds Service References for the bound services.

SERVICEOBJECTS

```
public static final CollectionType SERVICEOBJECTS
```

The serviceobjects collection type is used to indicate the collection holds Component Service Objects for the bound services.

PROPERTIES

```
public static final CollectionType PROPERTIES
```

The properties collection type is used to indicate the collection holds unmodifiable Maps containing the service properties of the bound services.

The Maps must implement `Comparable` with the `compareTo` method comparing service property maps using the same ordering as `ServiceReference.compareTo` based upon service ranking and service id.

TUPLE

```
public static final CollectionType TUPLE
```

The tuple collection type is used to indicate the collection holds unmodifiable Map.Entries whose key is an unmodifiable Map containing the service properties of the bound service, as specified in `PROPERTIES`, and whose value is the bound service object.

The Map.Entries must implement `Comparable` with the `compareTo` method comparing service property maps using the same ordering as `ServiceReference.compareTo` based upon service ranking and service id.

Method Detail

values

```
public static CollectionType[] values()
```

valueOf

```
public static CollectionType valueOf(String name)
```

toString

```
public String toString()
```

Overrides:

`toString` in class `Enum`

Annotation Type Component

org.osgi.service.component.annotations

```
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.TYPE)
@org.osgi.annotation.bundle.Requirement(namespace="osgi.extender",
            name="osgi.component",
            version="1.4.0")
public @interface Component
```

Identify the annotated class as a Service Component.

The annotated class is the implementation class of the Component.

This annotation is not processed at runtime by Service Component Runtime. It must be processed by tools and used to add a Component Description to the bundle.

See Also:

"The component element of a Component Description."

Field Summary		Page
String	**NAME** Special string representing the name of this Component.	37

Required Element Summary		Page
String[]	**configurationPid** The configuration PIDs for the configuration of this Component.	40
ConfigurationPolicy	**configurationPolicy** The configuration policy of this Component.	39
boolean	**enabled** Declares whether this Component is enabled when the bundle declaring it is started.	38
String	**factory** The factory identifier of this Component.	38
String[]	**factoryProperties** Factory property entries for this Factory Component.	41
String[]	**factoryProperty** Factory properties for this Factory Component.	41
boolean	**immediate** Declares whether this Component must be immediately activated upon becoming satisfied or whether activation should be delayed.	38
String	**name** The name of this Component.	37
String[]	**properties** Property entries for this Component.	39
String[]	**property** Properties for this Component.	39
Reference[]	**reference** The lookup strategy references of this Component.	41
ServiceScope	**scope** The service scope for the service of this Component.	40

Class<?>[]	**service** The types under which to register this Component as a service.	37
boolean	**servicefactory** **Deprecated.** Since 1.3.	38
String	**xmlns** The XML name space of the Component Description for this Component.	39

Field Detail

NAME

```
public static final String NAME = "$"
```

Special string representing the name of this Component.

This string can be used in `configurationPid()` to specify the name of the component as a configuration PID. For example:

```
@Component(configurationPid={"com.acme.system", Component.NAME})
```

Tools creating a Component Description from this annotation must replace the special string with the actual name of this Component.

Since:
1.3

Element Detail

name

```
public abstract String name
```

The name of this Component.

If not specified, the name of this Component is the fully qualified type name of the class being annotated.

Default:
""

See Also:
"The name attribute of the component element of a Component Description."

---

service

```
public abstract Class<?>[] service
```

The types under which to register this Component as a service.

If no service should be registered, the empty value `{}` must be specified.

If not specified, the service types for this Component are all the *directly* implemented interfaces of the class being annotated.

Default:
{}

See Also:
"The service element of a Component Description."

---

factory

`public abstract String` **`factory`**

The factory identifier of this Component. Specifying a factory identifier makes this Component a Factory Component.

If not specified, the default is that this Component is not a Factory Component.

Default:
""

See Also:
"The factory attribute of the component element of a Component Description."

---

servicefactory

`public abstract boolean` **`servicefactory`**

Deprecated. *Declares whether this Component uses the OSGi ServiceFactory concept and each bundle using this Component's service will receive a different component instance.*

This element is ignored when the `scope()` element does not have the default value. If `true`, this Component uses `bundle` service scope. If `false` or not specified, this Component uses `singleton` service scope. If the `factory()` element is specified or the `immediate()` element is specified with `true`, this element can only be specified with `false`.

Declares whether this Component uses the OSGi ServiceFactory concept and each bundle using this Component's service will receive a different component instance.

This element is ignored when the `scope()` element does not have the default value. If `true`, this Component uses `bundle` service scope. If `false` or not specified, this Component uses `singleton` service scope. If the `factory()` element is specified or the `immediate()` element is specified with `true`, this element can only be specified with `false`.

Default:
false
See Also:
"The scope attribute of the service element of a Component Description."

---

enabled

`public abstract boolean` **`enabled`**

Declares whether this Component is enabled when the bundle declaring it is started.

If `true` or not specified, this Component is enabled. If `false`, this Component is disabled.

Default:
true
See Also:
"The enabled attribute of the component element of a Component Description."

---

immediate

`public abstract boolean` **`immediate`**

Declares whether this Component must be immediately activated upon becoming satisfied or whether activation should be delayed.

If `true`, this Component must be immediately activated upon becoming satisfied. If `false`, activation of this Component is delayed. If this property is specified, its value must be `false` if the `factory()` property is also specified or must be `true` if the `service()` property is specified with an empty value.

If not specified, the default is `false` if the `factory()` property is specified or the `service()` property is not specified or specified with a non-empty value and `true` otherwise.

Default:
false
See Also:
"The immediate attribute of the component element of a Component Description."

---

property
`public abstract String[] property`

Properties for this Component.

Each property string is specified as `"name=value"`. The type of the property value can be specified in the name as `name:type=value`. The type must be one of the property types supported by the `type` attribute of the `property` element of a Component Description.

To specify a property with multiple values, use multiple name, value pairs. For example, `{"foo=bar", "foo=baz"}`.

Default:
{}
See Also:
"The property element of a Component Description."

---

properties
`public abstract String[] properties`

Property entries for this Component.

Specifies the name of an entry in the bundle whose contents conform to a standard Java Properties File. The entry is read and processed to obtain the properties and their values.

Default:
{}
See Also:
"The properties element of a Component Description."

---

xmlns
`public abstract String xmlns`

The XML name space of the Component Description for this Component.

If not specified, the XML name space of the Component Description for this Component should be the lowest Declarative Services XML name space which supports all the specification features used by this Component.

Default:
""
See Also:
"The XML name space specified for a Component Description."

---

configurationPolicy
`public abstract ConfigurationPolicy configurationPolicy`

The configuration policy of this Component.

Controls whether component configurations must be satisfied depending on the presence of a corresponding Configuration object in the OSGi Configuration Admin service. A corresponding configuration is a Configuration object where the PID equals the name of the component.

If not specified, the configuration policy is based upon whether the component is also annotated with the Meta Type `Designate` annotation.

- Not annotated with `Designate` - The configuration policy is `OPTIONAL`.
- Annotated with `Designate(factory=false)` - The configuration policy is `OPTIONAL`.
- Annotated with `Designate(factory=true)` - The configuration policy is `REQUIRE`.

Default:
ConfigurationPolicy.OPTIONAL
Since:
1.1
See Also:
"The configuration-policy attribute of the component element of a Component Description."

---

configurationPid
`public abstract String[] configurationPid`

The configuration PIDs for the configuration of this Component.

Each value specifies a configuration PID for this Component.

If no value is specified, the name of this Component is used as the configuration PID of this Component.

A special string (`"$"`) can be used to specify the name of the component as a configuration PID. The `NAME` constant holds this special string. For example:

```
@Component(configurationPid={"com.acme.system", Component.NAME})
```

Tools creating a Component Description from this annotation must replace the special string with the actual name of this Component.

Default:
{ "$" }
Since:
1.2
See Also:
"The configuration-pid attribute of the component element of a Component Description."

---

scope
`public abstract ServiceScope scope`

The service scope for the service of this Component.

If not specified (and the deprecated `servicefactory()` element is not specified), the `singleton` service scope is used. If the `factory()` element is specified or the `immediate()` element is specified with `true`, this element can only be specified with the `singleton` service scope.

Default:
ServiceScope.DEFAULT
Since:
1.3
See Also:
"The scope attribute of the service element of a Component Description."

---

reference

```
public abstract Reference[] reference
```

The lookup strategy references of this Component.

To access references using the lookup strategy, `Reference` annotations are specified naming the reference and declaring the type of the referenced service. The referenced service can be accessed using one of the `locateService` methods of `ComponentContext`.

To access references using method injection, bind methods are annotated with `Reference`. To access references using field injection, fields are annotated with `Reference`. To access references using constructor injection, constructor parameters are annotated with `Reference`.

Default:
{}
Since:
1.3
See Also:
"The reference element of a Component Description."

---

factoryProperty

```
public abstract String[] factoryProperty
```

Factory properties for this Factory Component.

Each factory property string is specified as `"name=value"`. The type of the factory property value can be specified in the name as `name:type=value`. The type must be one of the factory property types supported by the `type` attribute of the `factory-property` element of a Component Description.

To specify a factory property with multiple values, use multiple name, value pairs. For example, `{"foo=bar", "foo=baz"}`.

If specified, the `factory()` element must also be specified to indicate the component is a Factory Component.

Default:
{}
Since:
1.4
See Also:
"The factory-property element of a Component Description."

---

factoryProperties

```
public abstract String[] factoryProperties
```

Factory property entries for this Factory Component.

Specifies the name of an entry in the bundle whose contents conform to a standard Java Properties File. The entry is read and processed to obtain the factory properties and their values.

If specified, the `factory()` element must also be specified to indicate the component is a Factory Component.

Default:
{}
Since:
1.4
See Also:
"The factory-properties element of a Component Description."

Annotation Type ComponentPropertyType

org.osgi.service.component.annotations

```
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.ANNOTATION_TYPE)
public @interface ComponentPropertyType
```

Identify the annotated annotation as a Component Property Type.

Component Property Types can be applied as annotations to the implementation class of the Component. They can also be used as activation objects which means they can be used as parameter types for the component's constructor and life cycle methods `Activate`, `Deactivate`, and `Modified` as well as activation fields.

Component Property Types do not have to be annotated with this annotation to be used as parameter types but they must be annotated with this annotation to be used as annotations on the implementation class of the Component.

This annotation is not processed at runtime by Service Component Runtime. It must be processed by tools and used to add a Component Description to the bundle.

Since:

1.4

See Also:

"Component Property Types."

Enum ConfigurationPolicy

org.osgi.service.component.annotations

```
java.lang.Object
  └ java.lang.Enum<ConfigurationPolicy>
      └ org.osgi.service.component.annotations.ConfigurationPolicy
```

All Implemented Interfaces:

Comparable<ConfigurationPolicy>, Serializable

```
public enum ConfigurationPolicy
extends Enum<ConfigurationPolicy>
```

Configuration Policy for the `Component` annotation.

Controls whether component configurations must be satisfied depending on the presence of a corresponding Configuration object in the OSGi Configuration Admin service. A corresponding configuration is a Configuration object where the PID is the name of the component.

Since:

1.1

Enum Constant Summary	Page
IGNORE Always allow the component configuration to be satisfied and do not use the corresponding Configuration object even if it is present.	44
OPTIONAL Use the corresponding Configuration object if present but allow the component to be satisfied even if the corresponding Configuration object is not present.	43
REQUIRE There must be a corresponding Configuration object for the component configuration to become satisfied.	43

Method Summary		Page
String	**toString**()	44
static ConfigurationPolicy	**valueOf**(String name)	44
static ConfigurationPolicy[]	**values**()	44

Enum Constant Detail

OPTIONAL

```
public static final ConfigurationPolicy OPTIONAL
```

Use the corresponding Configuration object if present but allow the component to be satisfied even if the corresponding Configuration object is not present.

REQUIRE

```
public static final ConfigurationPolicy REQUIRE
```

There must be a corresponding Configuration object for the component configuration to become satisfied.

IGNORE

```
public static final ConfigurationPolicy IGNORE
```

Always allow the component configuration to be satisfied and do not use the corresponding Configuration object even if it is present.

Method Detail

values

```
public static ConfigurationPolicy[] values()
```

valueOf

```
public static ConfigurationPolicy valueOf(String name)
```

toString

```
public String toString()
```

Overrides:

`toString` in class `Enum`

Annotation Type Deactivate

org.osgi.service.component.annotations

```
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.METHOD)
public @interface Deactivate
```

Identify the annotated method as the `deactivate` method of a Service Component.

The annotated method is the deactivate method of the Component.

This annotation is not processed at runtime by Service Component Runtime. It must be processed by tools and used to add a Component Description to the bundle.

Since:

1.1

See Also:

"The deactivate attribute of the component element of a Component Description."

Enum FieldOption

org.osgi.service.component.annotations

```
java.lang.Object
  └java.lang.Enum<FieldOption>
      └org.osgi.service.component.annotations.FieldOption
```

All Implemented Interfaces:

Comparable<FieldOption>, Serializable

```
public enum FieldOption
extends Enum<FieldOption>
```

Field options for the `Reference` annotation.

Since:

1.3

Enum Constant Summary	Page
REPLACE The replace field option is used to replace the field value with a new value when there are changes to the bound services.	46
UPDATE The update field option is used to update the collection referenced by the field when there are changes to the bound services.	46

Method Summary		Page
String	**toString**()	47
static FieldOption	**valueOf**(String name)	47
static FieldOption[]	**values**()	47

Enum Constant Detail

UPDATE

```
public static final FieldOption UPDATE
```

The update field option is used to update the collection referenced by the field when there are changes to the bound services.

This field option can only be used when the field reference has dynamic policy and multiple cardinality.

REPLACE

```
public static final FieldOption REPLACE
```

The replace field option is used to replace the field value with a new value when there are changes to the bound services.

Method Detail

values

```
public static FieldOption[] values()
```

valueOf

```
public static FieldOption valueOf(String name)
```

toString

```
public String toString()
```

Overrides:

`toString` in class `Enum`

Annotation Type Modified

org.osgi.service.component.annotations

```
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.METHOD)
public @interface Modified
```

Identify the annotated method as the `modified` method of a Service Component.

The annotated method is the modified method of the Component.

This annotation is not processed at runtime by Service Component Runtime. It must be processed by tools and used to add a Component Description to the bundle.

Since:

1.1

See Also:

"The modified attribute of the component element of a Component Description."

Annotation Type Reference

org.osgi.service.component.annotations

```
@Retention(value=RetentionPolicy.CLASS)
@Target(value={
  ElementType.METHOD,
  ElementType.FIELD,
  ElementType.PARAMETER
})
public @interface Reference
```

Identify the annotated member or parameter as a reference of a Service Component.

When the annotation is applied to a method, the method is the bind method of the reference.

When the annotation is applied to a field, the field will contain the bound service(s) of the reference.

When the annotation is applied to a parameter of a constructor, the parameter will contain the bound service(s) of the reference.

This annotation is not processed at runtime by Service Component Runtime. It must be processed by tools and used to add a Component Description to the bundle.

In the generated Component Description for a component, the references must be ordered in ascending lexicographical order (using `String.compareTo`) of the reference name`s`.

See Also:

"The reference element of a Component Description."

Required Element Summary		Page
String	**bind** The name of the bind method for this reference.	52
ReferenceCardinality	**cardinality** The cardinality of this reference.	51
CollectionType	**collectionType** The collection type for this reference.	55
String	**field** The name of the field for this reference.	53
FieldOption	**fieldOption** The field option for this reference.	54
String	**name** The name of this reference.	50
int	**parameter** The zero-based parameter number of the constructor parameter for this reference.	54
ReferencePolicy	**policy** The policy for this reference.	51
ReferencePolicyOption	**policyOption** The policy option for this reference.	51
ReferenceScope	**scope** The reference scope for this reference.	52
Class<?>	**service** The type of the service for this reference.	50

String	**target** The target property for this reference.	51
String	**unbind** The name of the unbind method for this reference.	53
String	**updated** The name of the updated method for this reference.	52

Element Detail

name

```
public abstract String name
```

The name of this reference.

The name of this reference must be specified when using this annotation in the `Component.reference()` element since there is no annotated member from which the name can be determined. If not specified, the name of this reference is based upon how this annotation is used:

- Annotated method - If the method name begins with `bind`, `set` or `add`, that prefix is removed to create the name of the reference. Otherwise, the name of the reference is the method name.
- Annotated field - The name of the reference is the field name.
- Annotated constructor parameter - The name of the reference is the parameter name.

Default:
""

See Also:
"The name attribute of the reference element of a Component Description."

---

service

```
public abstract Class<?> service
```

The type of the service for this reference.

The type of the service for this reference must be specified when using this annotation in the `Component.reference()` element since there is no annotated member from which the type of the service can be determined.

If not specified, the type of the service for this reference is based upon how this annotation is used:

- Annotated method - The type of the service is the type of the first parameter of the method.
- Annotated field - The type of the service is based upon the type of the field being annotated and the cardinality of the reference. If the cardinality is either `0..n`, or `1..n`, the type of the field must be one of `java.util.Collection`, `java.util.List`, or a subtype of `java.util.Collection` so the type of the service is the generic type of the collection. Otherwise, the type of the service is the type of the field.
- Annotated constructor parameter - The type of the service is based upon the type of the parameter being annotated and the cardinality of the reference. If the cardinality is either `0..n`, or `1..n`, the type of the parameter must be one of `java.util.Collection`, `java.util.List`, or a subtype of `java.util.Collection` so the type of the service is the generic type of the collection. Otherwise, the type of the service is the type of the parameter.

Default:
Object.class

See Also:
"The interface attribute of the reference element of a Component Description."

---

cardinality

`public abstract ReferenceCardinality cardinality`

The cardinality of this reference.

If not specified, the cardinality of this reference is based upon how this annotation is used:

- Annotated method - The cardinality is `1..1`.
- Annotated field - The cardinality is based on the type of the field. If the type is either `java.util.Collection`, `java.util.List`, or a subtype of `java.util.Collection`, the cardinality is `0..n`. Otherwise the cardinality is `1..1`.
- `Component.reference()` element - The cardinality is `1..1`.
- Annotated constructor parameter - The cardinality is based on the type of the parameter. If the type is either `java.util.Collection`, `java.util.List`, or a subtype of `java.util.Collection`, the cardinality is `0..n`. Otherwise the cardinality is `1..1`.
- `Component.reference()` element - The cardinality is `1..1`.

Default:
ReferenceCardinality.MANDATORY
See Also:
"The cardinality attribute of the reference element of a Component Description."

---

policy

`public abstract ReferencePolicy policy`

The policy for this reference.

If not specified, the policy of this reference is based upon how this annotation is used:

- Annotated method - The policy is `STATIC`.
- Annotated field - The policy is based on the modifiers of the field. If the field is declared `volatile`, the policy is `ReferencePolicy.DYNAMIC`. Otherwise the policy is `STATIC`.
- Annotated constructor parameter - The policy is `STATIC`. STATIC policy must be used for constructor parameters.
- `Component.reference()` element - The policy is `STATIC`.

Default:
ReferencePolicy.STATIC
See Also:
"The policy attribute of the reference element of a Component Description."

---

target

`public abstract String target`

The target property for this reference.

If not specified, no target property is set.

Default:
""
See Also:
"The target attribute of the reference element of a Component Description."

---

policyOption

`public abstract ReferencePolicyOption policyOption`

The policy option for this reference.

If not specified, the `RELUCTANT` reference policy option is used.

Default:
ReferencePolicyOption.RELUCTANT
Since:
1.2
See Also:
"The policy-option attribute of the reference element of a Component Description."

---

scope
`public abstract ReferenceScope scope`

The reference scope for this reference.

If not specified, the `bundle` reference scope is used.

Default:
ReferenceScope.BUNDLE
Since:
1.3
See Also:
"The scope attribute of the reference element of a Component Description."

---

bind
`public abstract String bind`

The name of the bind method for this reference.

If specified and this reference annotates a method, the specified name must match the name of the annotated method.

If not specified, the name of the bind method is based upon how this annotation is used:

- Annotated method - The name of the annotated method is the name of the bind method.
- Annotated field - There is no bind method name.
- Annotated constructor parameter - There is no bind method name.
- `Component.reference()` element - There is no bind method name.

If there is a bind method name, the component must contain a method with that name.

Default:
""
Since:
1.3
See Also:
"The bind attribute of the reference element of a Component Description."

---

updated
`public abstract String updated`

The name of the updated method for this reference.

If not specified, the name of the updated method is based upon how this annotation is used:

- Annotated method - The name of the updated method is created from the name of the annotated method. If the name of the annotated method begins with `bind`, `set` or `add`, that prefix is replaced with `updated` to create the name candidate for the updated method. Otherwise, `updated` is prefixed to the name of the annotated method to create the name candidate for the updated method. If the component type contains a method with the candidate name, the candidate name is used as the name of the updated method. To declare no updated method when the component type contains a method with the candidate name, the value `"-"` must be used.
- Annotated field - There is no updated method name.
- Annotated constructor parameter - There is no updated method name.
- `Component.reference()` element - There is no updated method name.

If there is an updated method name, the component must contain a method with that name.

Default:

""

Since:

1.2

See Also:

"The updated attribute of the reference element of a Component Description."

---

unbind

```
public abstract String unbind
```

The name of the unbind method for this reference.

If not specified, the name of the unbind method is based upon how this annotation is used:

- Annotated method - The name of the unbind method is created from the name of the annotated method. If the name of the annotated method begins with `bind`, `set` or `add`, that prefix is replaced with `unbind`, `unset` or `remove`, respectively, to create the name candidate for the unbind method. Otherwise, `un` is prefixed to the name of the annotated method to create the name candidate for the unbind method. If the component type contains a method with the candidate name, the candidate name is used as the name of the unbind method. To declare no unbind method when the component type contains a method with the candidate name, the value `"-"` must be used.
- Annotated field - There is no unbind method name.
- Annotated constructor parameter - There is no unbind method name.
- `Component.reference()` element - There is no unbind method name.

If there is an unbind method name, the component must contain a method with that name.

Default:

""

See Also:

"The unbind attribute of the reference element of a Component Description."

---

field

```
public abstract String field
```

The name of the field for this reference.

If specified and this reference annotates a field, the specified name must match the name of the annotated field.

If not specified, the name of the field is based upon how this annotation is used:

- Annotated method - There is no field name.
- Annotated field - The name of the annotated field is the name of the field.
- Annotated constructor parameter - There is no field name.

- `Component.reference()` element - There is no field name.

If there is a field name, the component must contain a field with that name.

Default:
""

Since:
1.3

See Also:
"The field attribute of the reference element of a Component Description."

---

fieldOption

```
public abstract FieldOption fieldOption
```

The field option for this reference.

If not specified, the field option is based upon how this annotation is used:

- Annotated method - There is no field option.
- Annotated field - The field option is based upon the policy and cardinality of the reference and the modifiers of the field. If the policy is `ReferencePolicy.DYNAMIC`, the cardinality is `0..n` or `1..n`, and the field is declared `final`, the field option is `FieldOption.UPDATE`. Otherwise, the field option is `FieldOption.REPLACE`.
- Annotated constructor parameter - There is no field option.
- `Component.reference()` element - There is no field option.

Default:
FieldOption.REPLACE

Since:
1.3

See Also:
"The field-option attribute of the reference element of a Component Description."

---

parameter

```
public abstract int parameter
```

The zero-based parameter number of the constructor parameter for this reference.

If specified and this reference annotates an constructor parameter, the specified value must match the zero-based parameter number of the annotated constructor parameter.

If not specified, the parameter number is based upon how this annotation is used:

- Annotated method - There is no parameter number.
- Annotated field - There is no parameter number.
- Annotated constructor parameter - The zero-based parameter number of the parameter.
- `Component.reference()` element - There is no parameter number.

If there is a parameter number, the component must declare a constructor that has a parameter having the zero-based parameter number.

Default:
0

Since:
1.4

See Also:
"The parameter attribute of the reference element of a Component Description.", "The init attribute of the component element of a Component Description."

collectionType

`public abstract CollectionType` **`collectionType`**

The collection type for this reference.

If not specified, the collection type is based upon how this annotation is used:

- Annotated method - There is no collection type.
- Annotated field - The collection type is based upon the cardinality of the reference and the generic type of the field. If the cardinality is `0..n` or `1..n`, the collection type is inferred from the generic type of the list or collection. Otherwise, there is no collection type
- Annotated constructor method parameter - The collection type is based upon the cardinality of the reference and the generic type of the parameter. If the cardinality is `0..n` or `1..n`, the collection type is inferred from the generic type of the list or collection. Otherwise, there is no collection type
- `Component.reference()` element - There is no collection type.

Default:

CollectionType.SERVICE

Since:

1.4

See Also:

"The field-collection-type attribute of the reference element of a Component Description."

Enum ReferenceCardinality

org.osgi.service.component.annotations

```
java.lang.Object
  └java.lang.Enum<ReferenceCardinality>
      └org.osgi.service.component.annotations.ReferenceCardinality
```

All Implemented Interfaces:

Comparable<ReferenceCardinality>, Serializable

```
public enum ReferenceCardinality
extends Enum<ReferenceCardinality>
```

Cardinality for the `Reference` annotation.

Specifies if the reference is optional and if the component implementation support a single bound service or multiple bound services.

Enum Constant Summary	Page
AT_LEAST_ONE The reference is mandatory and multiple.	57
MANDATORY The reference is mandatory and unary.	56
MULTIPLE The reference is optional and multiple.	56
OPTIONAL The reference is optional and unary.	56

Method Summary		Page
String	**toString**()	57
static ReferenceCardinality	**valueOf**(String name)	57
static ReferenceCardinality[]	**values**()	57

Enum Constant Detail

OPTIONAL

```
public static final ReferenceCardinality OPTIONAL
```

The reference is optional and unary. That is, the reference has a cardinality of `0..1`.

MANDATORY

```
public static final ReferenceCardinality MANDATORY
```

The reference is mandatory and unary. That is, the reference has a cardinality of `1..1`.

MULTIPLE

```
public static final ReferenceCardinality MULTIPLE
```

The reference is optional and multiple. That is, the reference has a cardinality of `0..n`.

AT_LEAST_ONE

```
public static final ReferenceCardinality AT_LEAST_ONE
```

The reference is mandatory and multiple. That is, the reference has a cardinality of `1..n`.

Method Detail

values

```
public static ReferenceCardinality[] values()
```

valueOf

```
public static ReferenceCardinality valueOf(String name)
```

toString

```
public String toString()
```

Overrides:

`toString` in class `Enum`

Enum ReferencePolicy

org.osgi.service.component.annotations

```
java.lang.Object
  └java.lang.Enum<ReferencePolicy>
      └org.osgi.service.component.annotations.ReferencePolicy
```

All Implemented Interfaces:

Comparable<ReferencePolicy>, Serializable

```
public enum ReferencePolicy
extends Enum<ReferencePolicy>
```

Policy for the `Reference` annotation.

Enum Constant Summary	Page
DYNAMIC The dynamic policy is slightly more complex since the component implementation must properly handle changes in the set of bound services.	58
STATIC The static policy is the most simple policy and is the default policy.	58

Method Summary		Page
String	**toString**()	59
static ReferencePolicy	**valueOf**(String name)	59
static ReferencePolicy[]	**values**()	59

Enum Constant Detail

STATIC

```
public static final ReferencePolicy STATIC
```

The static policy is the most simple policy and is the default policy. A component instance never sees any of the dynamics. Component configurations are deactivated before any bound service for a reference having a static policy becomes unavailable. If a target service is available to replace the bound service which became unavailable, the component configuration must be reactivated and bound to the replacement service.

DYNAMIC

```
public static final ReferencePolicy DYNAMIC
```

The dynamic policy is slightly more complex since the component implementation must properly handle changes in the set of bound services. With the dynamic policy, SCR can change the set of bound services without deactivating a component configuration. If the component uses method injection to access services, then the component instance will be notified of changes in the set of bound services by calls to the bind and unbind methods.

Method Detail

values

```
public static ReferencePolicy[] values()
```

valueOf

```
public static ReferencePolicy valueOf(String name)
```

toString

```
public String toString()
```

Overrides:

`toString` in class `Enum`

Enum ReferencePolicyOption

org.osgi.service.component.annotations

```
java.lang.Object
  └java.lang.Enum<ReferencePolicyOption>
      └org.osgi.service.component.annotations.ReferencePolicyOption
```

All Implemented Interfaces:

Comparable<ReferencePolicyOption>, Serializable

```
public enum ReferencePolicyOption
extends Enum<ReferencePolicyOption>
```

Policy option for the `Reference` annotation.

Since:

1.2

Enum Constant Summary	Page
GREEDY The greedy policy option is a valid policy option for both `static` and `dynamic` reference policies.	60
RELUCTANT The reluctant policy option is the default policy option for both `static` and `dynamic` reference policies.	60

Method Summary		Page
String	**toString**()	61
static ReferencePolicyOption	**valueOf**(String name)	61
static ReferencePolicyOption[]	**values**()	61

Enum Constant Detail

RELUCTANT

```
public static final ReferencePolicyOption RELUCTANT
```

The reluctant policy option is the default policy option for both `static` and `dynamic` reference policies. When a new target service for a reference becomes available, references having the reluctant policy option for the static policy or the dynamic policy with a unary cardinality will ignore the new target service. References having the dynamic policy with a multiple cardinality will bind the new target service.

GREEDY

```
public static final ReferencePolicyOption GREEDY
```

The greedy policy option is a valid policy option for both `static` and `dynamic` reference policies. When a new target service for a reference becomes available, references having the greedy policy option will bind the new target service.

Method Detail

values

```
public static ReferencePolicyOption[] values()
```

valueOf

```
public static ReferencePolicyOption valueOf(String name)
```

toString

```
public String toString()
```

Overrides:

`toString` in class `Enum`

Enum ReferenceScope

org.osgi.service.component.annotations

```
java.lang.Object
  └ java.lang.Enum<ReferenceScope>
      └ org.osgi.service.component.annotations.ReferenceScope
```

All Implemented Interfaces:

Comparable<ReferenceScope>, Serializable

```
public enum ReferenceScope
extends Enum<ReferenceScope>
```

Reference scope for the `Reference` annotation.

Since:

1.3

Enum Constant Summary	Page
BUNDLE A single service object is used for all references to the service in this bundle.	62
PROTOTYPE If the bound service has prototype service scope, then each instance of the component with this reference can receive a unique instance of the service.	62
PROTOTYPE_REQUIRED Bound services must have prototype service scope.	62

Method Summary		Page
String	**toString**()	63
static ReferenceScope	**valueOf**(String name)	63
static ReferenceScope[]	**values**()	63

Enum Constant Detail

BUNDLE

```
public static final ReferenceScope BUNDLE
```

A single service object is used for all references to the service in this bundle.

PROTOTYPE

```
public static final ReferenceScope PROTOTYPE
```

If the bound service has prototype service scope, then each instance of the component with this reference can receive a unique instance of the service. If the bound service does not have prototype service scope, then this reference scope behaves the same as `BUNDLE`.

PROTOTYPE_REQUIRED

```
public static final ReferenceScope PROTOTYPE_REQUIRED
```

Bound services must have prototype service scope. Each instance of the component with this reference can receive a unique instance of the service.

Method Detail

values

```
public static ReferenceScope[] values()
```

valueOf

```
public static ReferenceScope valueOf(String name)
```

toString

```
public String toString()
```

Overrides:

`toString` in class `Enum`

Enum ServiceScope

org.osgi.service.component.annotations

```
java.lang.Object
  └ java.lang.Enum<ServiceScope>
      └ org.osgi.service.component.annotations.ServiceScope
```

All Implemented Interfaces:

Comparable<ServiceScope>, Serializable

```
public enum ServiceScope
extends Enum<ServiceScope>
```

Service scope for the `Component` annotation.

Since:

1.3

Enum Constant Summary	Page
BUNDLE When the component is registered as a service, it must be registered as a bundle scope service and an instance of the component must be created for each bundle using the service.	64
DEFAULT Default element value for annotation.	65
PROTOTYPE When the component is registered as a service, it must be registered as a prototype scope service and an instance of the component must be created for each distinct request for the service.	65
SINGLETON When the component is registered as a service, it must be registered as a bundle scope service but only a single instance of the component must be used for all bundles using the service.	64

Method Summary		Page
String	**toString**()	65
static ServiceScope	**valueOf**(String name)	65
static ServiceScope[]	**values**()	65

Enum Constant Detail

SINGLETON

```
public static final ServiceScope SINGLETON
```

When the component is registered as a service, it must be registered as a bundle scope service but only a single instance of the component must be used for all bundles using the service.

BUNDLE

```
public static final ServiceScope BUNDLE
```

When the component is registered as a service, it must be registered as a bundle scope service and an instance of the component must be created for each bundle using the service.

PROTOTYPE

```
public static final ServiceScope PROTOTYPE
```

When the component is registered as a service, it must be registered as a prototype scope service and an instance of the component must be created for each distinct request for the service.

DEFAULT

```
public static final ServiceScope DEFAULT
```

Default element value for annotation. This is used to distinguish the default value for an element and should not otherwise be used.

Method Detail

values

```
public static ServiceScope[] values()
```

valueOf

```
public static ServiceScope valueOf(String name)
```

toString

```
public String toString()
```

Overrides:

`toString` in class `Enum`

Package org.osgi.service.component.propertytypes

```
@org.osgi.annotation.versioning.Version(value="1.4.0")
```

Component Property Types Package Version 1.4.

See:

[Description](#)

Annotation Types Summary		Page
ExportedService	Component Property Type for the remote service properties for an exported service.	67
ServiceDescription	Component Property Type for the `service.description` service property.	69
ServiceRanking	Component Property Type for the `service.ranking` service property.	70
ServiceVendor	Component Property Type for the `service.vendor` service property.	71

Package org.osgi.service.component.propertytypes Description

Component Property Types Package Version 1.4.

When used as annotations, component property types are processed by tools to generate Component Descriptions which are used at runtime.

Bundles wishing to use this package at runtime must list the package in the Import-Package header of the bundle's manifest.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.service.component.propertytypes; version="[1.4,2.0)"
```

Annotation Type ExportedService

org.osgi.service.component.propertytypes

```
@ComponentPropertyType
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.TYPE)
public @interface ExportedService
```

Component Property Type for the remote service properties for an exported service.

This annotation can be used on a `Component` to declare the values of the remote service properties for an exported service.

Since:
1.4

See Also:
"Component Property Types", "Remote Services Specification"

Required Element Summary		Page
String[]	**service_exported_configs** Service property identifying the configuration types that should be used to export the service.	67
String[]	**service_exported_intents** Service property identifying the intents that the distribution provider must implement to distribute the service.	68
String[]	**service_exported_intents_extra** Service property identifying the extra intents that the distribution provider must implement to distribute the service.	68
Class<?>[]	**service_exported_interfaces** Service property marking the service for export.	67
String[]	**service_intents** Service property identifying the intents that the distribution provider must implement to distribute the service.	68

Element Detail

service_exported_interfaces

```
public abstract Class<?>[] service_exported_interfaces
```

Service property marking the service for export. It defines the interfaces under which the service can be exported.

If an empty array is specified, the property is not added to the component description.

Returns:
The exported service interfaces.

See Also:
`org.osgi.framework.Constants.SERVICE_EXPORTED_INTERFACES`

---

service_exported_configs

```
public abstract String[] service_exported_configs
```

Service property identifying the configuration types that should be used to export the service.

If an empty array is specified, the default value, the property is not added to the component description.

Default:
{}
Returns:
The configuration types.
See Also:
`org.osgi.framework.Constants.SERVICE_EXPORTED_CONFIGS`

---

service_exported_intents
`public abstract String[] service_exported_intents`

Service property identifying the intents that the distribution provider must implement to distribute the service.

If an empty array is specified, the default value, the property is not added to the component description.

Default:
{}
Returns:
The intents that the distribution provider must implement to distribute the service.
See Also:
`org.osgi.framework.Constants.SERVICE_EXPORTED_INTENTS`

---

service_exported_intents_extra
`public abstract String[] service_exported_intents_extra`

Service property identifying the extra intents that the distribution provider must implement to distribute the service.

If an empty array is specified, the default value, the property is not added to the component description.

Default:
{}
Returns:
The extra intents that the distribution provider must implement to distribute the service.
See Also:
`org.osgi.framework.Constants.SERVICE_EXPORTED_INTENTS_EXTRA`

---

service_intents
`public abstract String[] service_intents`

Service property identifying the intents that the distribution provider must implement to distribute the service.

If an empty array is specified, the default value, the property is not added to the component description.

Default:
{}
Returns:
The intents that the service implements.
See Also:
`org.osgi.framework.Constants.SERVICE_INTENTS`

Annotation Type ServiceDescription

org.osgi.service.component.propertytypes

```
@ComponentPropertyType
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.TYPE)
public @interface ServiceDescription
```

Component Property Type for the `service.description` service property.

This annotation can be used on a `Component` to declare the value of the `org.osgi.framework.Constants.SERVICE_DESCRIPTION` service property.

Since:
1.4

See Also:
"Component Property Types"

Required Element Summary		Page
String	**value** Service property identifying a service's description.	69

Element Detail

value

```
public abstract String value
```

Service property identifying a service's description.

Returns:
The service description.

See Also:
`org.osgi.framework.Constants.SERVICE_DESCRIPTION`

Annotation Type ServiceRanking

org.osgi.service.component.propertytypes

```
@ComponentPropertyType
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.TYPE)
public @interface ServiceRanking
```

Component Property Type for the `service.ranking` service property.

This annotation can be used on a `Component` to declare the value of the `org.osgi.framework.Constants.SERVICE_RANKING` service property.

Since:
1.4

See Also:
"Component Property Types"

Required Element Summary		Page
int	**value** Service property identifying a service's ranking.	70

Element Detail

value

```
public abstract int value
```

Service property identifying a service's ranking.

Returns:
The service ranking.

See Also:
`org.osgi.framework.Constants.SERVICE_RANKING`

Annotation Type ServiceVendor

org.osgi.service.component.propertytypes

```
@ComponentPropertyType
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.TYPE)
public @interface ServiceVendor
```

Component Property Type for the `service.vendor` service property.

This annotation can be used on a `Component` to declare the value of the `org.osgi.framework.Constants.SERVICE_VENDOR` service property.

Since:
1.4

See Also:
"Component Property Types"

Required Element Summary		Page
String	**value** Service property identifying a service's vendor.	71

Element Detail

value

```
public abstract String value
```

Service property identifying a service's vendor.

Returns:
The service vendor.

See Also:
`org.osgi.framework.Constants.SERVICE_VENDOR`

Package org.osgi.service.component.runtime

```
@org.osgi.annotation.versioning.Version(value="1.4.0")
```

Service Component Runtime Package Version 1.4.

See:

[Description](#)

Interface Summary		Page
ServiceComponentRuntime	The `ServiceComponentRuntime` service represents the Declarative Services actor, known as Service Component Runtime (SCR), that manages the service components and their life cycle.	73

Package org.osgi.service.component.runtime Description

Service Component Runtime Package Version 1.4.

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.component.runtime; version="[1.4,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.component.runtime; version="[1.4,1.5)"
```

Interface ServiceComponentRuntime

org.osgi.service.component.runtime

```
@org.osgi.annotation.versioning.ProviderType
public interface ServiceComponentRuntime
```

The `ServiceComponentRuntime` service represents the Declarative Services actor, known as Service Component Runtime (SCR), that manages the service components and their life cycle. The `ServiceComponentRuntime` service allows introspection of the components managed by Service Component Runtime.

This service differentiates between a `ComponentDescriptionDTO` and a `ComponentConfigurationDTO`. A `ComponentDescriptionDTO` is a representation of a declared component description. A `ComponentConfigurationDTO` is a representation of an actual instance of a declared component description parameterized by component properties.

This service must be registered with a `org.osgi.framework.Constants.SERVICE_CHANGECOUNT` service property that must be updated each time the SCR DTOs available from this service change.

Access to this service requires the `ServicePermission[ServiceComponentRuntime, GET]` permission. It is intended that only administrative bundles should be granted this permission to limit access to the potentially intrusive methods provided by this service.

Since:

1.3

ThreadSafe

Method Summary		Page
org.osgi.util.promise.Promise<Void>	**disableComponent**(ComponentDescriptionDTO description) Disables the specified component description.	75
org.osgi.util.promise.Promise<Void>	**enableComponent**(ComponentDescriptionDTO description) Enables the specified component description.	75
Collection<ComponentConfigurationDTO>	**getComponentConfigurationDTOs**(ComponentDescriptionDTO description) Returns the component configurations for the specified component description.	74
ComponentDescriptionDTO	**getComponentDescriptionDTO**(org.osgi.framework.Bundle bundle, String name) Returns the ComponentDescriptionDTO declared with the specified name by the specified bundle.	74
Collection<ComponentDescriptionDTO>	**getComponentDescriptionDTOs**(org.osgi.framework.Bundle... bundles) Returns the component descriptions declared by the specified active bundles.	73
boolean	**isComponentEnabled**(ComponentDescriptionDTO description) Returns whether the specified component description is currently enabled.	74

Method Detail

getComponentDescriptionDTOs

```
Collection<ComponentDescriptionDTO> getComponentDescriptionDTOs(org.osgi.framework.Bundle... bundles)
```

Returns the component descriptions declared by the specified active bundles.

Only component descriptions from active bundles are returned. If the specified bundles have no declared components or are not active, an empty collection is returned.

Parameters:

`bundles` - The bundles whose declared component descriptions are to be returned. Specifying no bundles, or the equivalent of an empty `Bundle` array, will return the declared component descriptions from all active bundles.

Returns:

The declared component descriptions of the specified active `bundles`. An empty collection is returned if there are no component descriptions for the specified active bundles.

---

getComponentDescriptionDTO

`ComponentDescriptionDTO` **`getComponentDescriptionDTO`**`(org.osgi.framework.Bundle bundle, String name)`

Returns the `ComponentDescriptionDTO` declared with the specified name by the specified bundle.

Only component descriptions from active bundles are returned. `null` if no such component is declared by the given `bundle` or the bundle is not active.

Parameters:

`bundle` - The bundle declaring the component description. Must not be `null`.

`name` - The name of the component description. Must not be `null`.

Returns:

The declared component description or `null` if the specified bundle is not active or does not declare a component description with the specified name.

---

getComponentConfigurationDTOs

`Collection<ComponentConfigurationDTO>` **`getComponentConfigurationDTOs`**`(ComponentDescriptionDTO description)`

Returns the component configurations for the specified component description.

Parameters:

`description` - The component description. Must not be `null`.

Returns:

A collection containing a snapshot of the current component configurations for the specified component description. An empty collection is returned if there are none or if the provided component description does not belong to an active bundle.

---

isComponentEnabled

`boolean` **`isComponentEnabled`**`(ComponentDescriptionDTO description)`

Returns whether the specified component description is currently enabled.

The enabled state of a component description is initially set by the `enabled` attribute of the component description.

Parameters:

`description` - The component description. Must not be `null`.

Returns:

`true` if the specified component description is currently enabled. Otherwise, `false`.

See Also:

`enableComponent(ComponentDescriptionDTO)`,
`disableComponent(ComponentDescriptionDTO)`,
`ComponentContext.disableComponent(String)`,
`ComponentContext.enableComponent(String)`

---

enableComponent

`org.osgi.util.promise.Promise<Void>` **`enableComponent`**`(ComponentDescriptionDTO description)`

Enables the specified component description.

If the specified component description is currently enabled, this method has no effect.

This method must return after changing the enabled state of the specified component description. Any actions that result from this, such as activating or deactivating a component configuration, must occur asynchronously to this method call.

Parameters:

`description` - The component description to enable. Must not be `null`.

Returns:

A promise that will be resolved when the actions that result from changing the enabled state of the specified component have completed. If the provided description does not belong to an active bundle, a failed promise is returned.

See Also:

`isComponentEnabled(ComponentDescriptionDTO)`

---

disableComponent

`org.osgi.util.promise.Promise<Void>` **`disableComponent`**`(ComponentDescriptionDTO description)`

Disables the specified component description.

If the specified component description is currently disabled, this method has no effect.

This method must return after changing the enabled state of the specified component description. Any actions that result from this, such as activating or deactivating a component configuration, must occur asynchronously to this method call.

Parameters:

`description` - The component description to disable. Must not be `null`.

Returns:

A promise that will be resolved when the actions that result from changing the enabled state of the specified component have completed. If the provided description does not belong to an active bundle, a failed promise is returned.

See Also:

`isComponentEnabled(ComponentDescriptionDTO)`

Package org.osgi.service.component.runtime.dto

```
@org.osgi.annotation.versioning.Version(value="1.4.0")
```

Service Component Runtime Data Transfer Objects Package Version 1.4.

See:

Description

Class Summary		Page
ComponentConfigurationDTO	A representation of an actual instance of a declared component description parameterized by component properties.	77
ComponentDescriptionDTO	A representation of a declared component description.	81
ReferenceDTO	A representation of a declared reference to a service.	85
SatisfiedReferenceDTO	A representation of a satisfied reference.	89
UnsatisfiedReferenceDTO	A representation of an unsatisfied reference.	91

Package org.osgi.service.component.runtime.dto Description

Service Component Runtime Data Transfer Objects Package Version 1.4.

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.component.runtime.dto; version="[1.4,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.component.runtime.dto; version="[1.4,1.5)"
```

Class ComponentConfigurationDTO

org.osgi.service.component.runtime.dto

```
java.lang.Object
  └org.osgi.dto.DTO
      └org.osgi.service.component.runtime.dto.ComponentConfigurationDTO
```

```
public class ComponentConfigurationDTO
extends org.osgi.dto.DTO
```

A representation of an actual instance of a declared component description parameterized by component properties.

Since:
1.3
NotThreadSafe

Field Summary		Page
static int	**ACTIVE** The component configuration is active.	78
ComponentDescriptionDTO	**description** The representation of the component configuration's component description.	78
static int	**FAILED_ACTIVATION** The component configuration failed to activate.	78
String	**failure** The failure information if the component configuration state is FAILED_ACTIVATION.	79
long	**id** The id of the component configuration.	79
Map<String,Object>	**properties** The component properties for the component configuration.	79
static int	**SATISFIED** The component configuration is satisfied.	78
SatisfiedReferenceDTO[]	**satisfiedReferences** The satisfied references.	79
org.osgi.framework.dto.ServiceReferenceDTO	**service** The registered service of the component configuration.	79
int	**state** The current state of the component configuration.	78
static int	**UNSATISFIED_CONFIGURATION** The component configuration is unsatisfied due to a missing required configuration.	78
static int	**UNSATISFIED_REFERENCE** The component configuration is unsatisfied due to an unsatisfied reference.	78
UnsatisfiedReferenceDTO[]	**unsatisfiedReferences** The unsatisfied references.	79

Constructor Summary	Page

ComponentConfigurationDTO()	80

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

UNSATISFIED_CONFIGURATION

```
public static final int UNSATISFIED_CONFIGURATION = 1
```

The component configuration is unsatisfied due to a missing required configuration.

UNSATISFIED_REFERENCE

```
public static final int UNSATISFIED_REFERENCE = 2
```

The component configuration is unsatisfied due to an unsatisfied reference.

SATISFIED

```
public static final int SATISFIED = 4
```

The component configuration is satisfied.

Any services declared by the component description are registered.

ACTIVE

```
public static final int ACTIVE = 8
```

The component configuration is active.

This is the normal operational state of a component configuration.

FAILED_ACTIVATION

```
public static final int FAILED_ACTIVATION = 16
```

The component configuration failed to activate.

This means the component configuration is satisfied but that either:

- an exception occurred loading the implementation class,
- the static initializer threw an exception,
- the constructor threw an exception, or
- the activate method threw an exception.

The failure information from the exception is available from failure.

Since:

1.4

description

```
public ComponentDescriptionDTO description
```

The representation of the component configuration's component description.

state

```
public int state
```

The current state of the component configuration.

This is one of UNSATISFIED_CONFIGURATION, UNSATISFIED_REFERENCE, SATISFIED, ACTIVE, or FAILED_ACTIVATION.

id

public long **id**

The id of the component configuration.

The id is a non-persistent, unique value assigned at runtime. The id is also available as the component.id component property. The value of this field is unspecified if the state of this component configuration is unsatisfied.

properties

public Map<String,Object> **properties**

The component properties for the component configuration.

See Also:

ComponentContext.getProperties()

satisfiedReferences

public SatisfiedReferenceDTO[] **satisfiedReferences**

The satisfied references.

Each SatisfiedReferenceDTO in the array represents a satisfied reference of the component configuration. The array must be empty if the component configuration has no satisfied references.

unsatisfiedReferences

public UnsatisfiedReferenceDTO[] **unsatisfiedReferences**

The unsatisfied references.

Each UnsatisfiedReferenceDTO in the array represents an unsatisfied reference of the component configuration. The array must be empty if the component configuration has no unsatisfied references.

failure

public String **failure**

The failure information if the component configuration state is FAILED_ACTIVATION.

This is the failure exception converted to a String using:

```
StringWriter sw = new StringWriter();
exception.printStackTrace(new PrintWriter(sw));
sw.toString();
```

This must be null if the component configuration state is not FAILED_ACTIVATION.

Since:

1.4

service

public org.osgi.framework.dto.ServiceReferenceDTO **service**

The registered service of the component configuration.

This must be non-null if the component configuration is registered as a service. Otherwise it must be null.

Since:

1.4

Constructor Detail

ComponentConfigurationDTO

```
public ComponentConfigurationDTO()
```

Class ComponentDescriptionDTO

org.osgi.service.component.runtime.dto

```
java.lang.Object
  └org.osgi.dto.DTO
      └org.osgi.service.component.runtime.dto.ComponentDescriptionDTO
```

```
public class ComponentDescriptionDTO
extends org.osgi.dto.DTO
```

A representation of a declared component description.

Since:
1.3
NotThreadSafe

Field Summary		Page
String	**activate** The name of the activate method.	83
String[]	**activationFields** The activation fields.	84
org.osgi.framework.dto.BundleDTO	**bundle** The bundle declaring the component description.	82
String[]	**configurationPid** The configuration pids.	84
String	**configurationPolicy** The configuration policy.	83
String	**deactivate** The name of the deactivate method.	83
boolean	**defaultEnabled** The initial enabled state.	82
String	**factory** The component factory name.	82
Map<String,Object>	**factoryProperties** The factory properties.	84
boolean	**immediate** The immediate state.	83
String	**implementationClass** The fully qualified name of the implementation class.	82
int	**init** The constructor parameter count.	84
String	**modified** The name of the modified method.	83
String	**name** The name of the component.	82
Map<String,Object>	**properties** The component properties.	83

ReferenceDTO[]	**references** The referenced services.	83
String	**scope** The service scope.	82
String[]	**serviceInterfaces** The fully qualified names of the service interfaces.	83

Constructor Summary	Page
ComponentDescriptionDTO()	84

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

name

```
public String name
```

The name of the component.

This is declared in the `name` attribute of the `component` element. This must be the default name if the component description does not declare a name.

bundle

```
public org.osgi.framework.dto.BundleDTO bundle
```

The bundle declaring the component description.

factory

```
public String factory
```

The component factory name.

This is declared in the `factory` attribute of the `component` element. This must be `null` if the component description is not declared as a factory component.

scope

```
public String scope
```

The service scope.

This is declared in the `scope` attribute of the `service` element. This must be `null` if the component description does not declare any service interfaces.

implementationClass

```
public String implementationClass
```

The fully qualified name of the implementation class.

This is declared in the `class` attribute of the `implementation` element.

defaultEnabled

```
public boolean defaultEnabled
```

The initial enabled state.

This is declared in the `enabled` attribute of the `component` element.

immediate

```
public boolean immediate
```

The immediate state.

This is declared in the `immediate` attribute of the `component` element.

serviceInterfaces

```
public String[] serviceInterfaces
```

The fully qualified names of the service interfaces.

These are declared in the `interface` attribute of the `provide` elements. The array must be empty if the component description does not declare any service interfaces.

properties

```
public Map<String,Object> properties
```

The component properties.

These are declared in the component description by the `property` and `properties` elements as well as the `target` attribute of the `reference` elements.

references

```
public ReferenceDTO[] references
```

The referenced services.

These are declared in the `reference` elements. The array must be empty if the component description does not declare references to any services.

activate

```
public String activate
```

The name of the activate method.

This is declared in the `activate` attribute of the `component` element. This must be `null` if the component description does not declare an activate method name.

deactivate

```
public String deactivate
```

The name of the deactivate method.

This is declared in the `deactivate` attribute of the `component` element. This must be `null` if the component description does not declare a deactivate method name.

modified

```
public String modified
```

The name of the modified method.

This is declared in the `modified` attribute of the `component` element. This must be `null` if the component description does not declare a modified method name.

configurationPolicy

```
public String configurationPolicy
```

The configuration policy.

This is declared in the `configuration-policy` attribute of the `component` element. This must be the default configuration policy if the component description does not declare a configuration policy.

configurationPid
`public String[] configurationPid`

The configuration pids.

These are declared in the `configuration-pid` attribute of the `component` element. This must contain the default configuration pid if the component description does not declare a configuration pid.

factoryProperties
`public Map<String,Object> factoryProperties`

The factory properties.

These are declared in the component description by the `factory-property` and `factory-properties` elements. This must be `null` if the component description is not declared as a `factory component`.

Since:
1.4

---

activationFields
`public String[] activationFields`

The activation fields.

These are declared in the `activation-fields` attribute of the `component` element. The array must be empty if the component description does not declare any activation fields.

Since:
1.4

---

init
`public int init`

The constructor parameter count.

This is declared in the `init` attribute of the `component` element. This must be `0` if the component description does not declare an `init` attribute.

Since:
1.4

Constructor Detail

ComponentDescriptionDTO
`public ComponentDescriptionDTO()`

Class ReferenceDTO

org.osgi.service.component.runtime.dto

```
java.lang.Object
  └org.osgi.dto.DTO
      └org.osgi.service.component.runtime.dto.ReferenceDTO
```

```
public class ReferenceDTO
extends org.osgi.dto.DTO
```

A representation of a declared reference to a service.

Since:
1.3
NotThreadSafe

Field Summary		Page
String	**bind** The name of the bind method of the reference.	86
String	**cardinality** The cardinality of the reference.	86
String	**collectionType** The collection type for the reference.	87
String	**field** The name of the field of the reference.	87
String	**fieldOption** The field option of the reference.	87
String	**interfaceName** The service interface of the reference.	86
String	**name** The name of the reference.	86
Integer	**parameter** The zero-based parameter number of the constructor parameter for the reference.	87
String	**policy** The policy of the reference.	86
String	**policyOption** The policy option of the reference.	86
String	**scope** The scope of the reference.	87
String	**target** The target of the reference.	86
String	**unbind** The name of the unbind method of the reference.	87
String	**updated** The name of the updated method of the reference.	87

Constructor Summary	Page
ReferenceDTO()	88

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

name

public String **name**

The name of the reference.

This is declared in the `name` attribute of the `reference` element. This must be the default name if the component description does not declare a name for the reference.

interfaceName

public String **interfaceName**

The service interface of the reference.

This is declared in the `interface` attribute of the `reference` element.

cardinality

public String **cardinality**

The cardinality of the reference.

This is declared in the `cardinality` attribute of the `reference` element. This must be the default cardinality if the component description does not declare a cardinality for the reference.

policy

public String **policy**

The policy of the reference.

This is declared in the `policy` attribute of the `reference` element. This must be the default policy if the component description does not declare a policy for the reference.

policyOption

public String **policyOption**

The policy option of the reference.

This is declared in the `policy-option` attribute of the `reference` element. This must be the default policy option if the component description does not declare a policy option for the reference.

target

public String **target**

The target of the reference.

This is declared in the `target` attribute of the `reference` element. This must be `null` if the component description does not declare a target for the reference.

bind

public String **bind**

The name of the bind method of the reference.

This is declared in the `bind` attribute of the `reference` element. This must be `null` if the component description does not declare a bind method for the reference.

unbind
`public String **unbind**`

The name of the unbind method of the reference.

This is declared in the `unbind` attribute of the `reference` element. This must be `null` if the component description does not declare an unbind method for the reference.

updated
`public String **updated**`

The name of the updated method of the reference.

This is declared in the `updated` attribute of the `reference` element. This must be `null` if the component description does not declare an updated method for the reference.

field
`public String **field**`

The name of the field of the reference.

This is declared in the `field` attribute of the `reference` element. This must be `null` if the component description does not declare a field for the reference.

fieldOption
`public String **fieldOption**`

The field option of the reference.

This is declared in the `field-option` attribute of the `reference` element. This must be `null` if the component description does not declare a field for the reference.

scope
`public String **scope**`

The scope of the reference.

This is declared in the `scope` attribute of the `reference` element. This must be the default scope if the component description does not declare a scope for the reference.

parameter
`public Integer **parameter**`

The zero-based parameter number of the constructor parameter for the reference.

This is declared in the `parameter` attribute of the `reference` element. This must be `null` if the component description does not declare a parameter number for the reference.

Since:
1.4

collectionType
`public String **collectionType**`

The collection type for the reference.

This is declared in the `field-collection-type` attribute of the `reference` element. This must be `null` if the component description does not declare a collection type for the reference.

Since:

1.4

Constructor Detail

ReferenceDTO

```
public ReferenceDTO()
```

Class SatisfiedReferenceDTO

org.osgi.service.component.runtime.dto

```
java.lang.Object
  └org.osgi.dto.DTO
      └org.osgi.service.component.runtime.dto.SatisfiedReferenceDTO
```

```
public class SatisfiedReferenceDTO
extends org.osgi.dto.DTO
```

A representation of a satisfied reference.

Since:
1.3
NotThreadSafe

Field Summary		Page
org.osgi.framework.dto.ServiceReferenceDTO[]	**boundServices** The bound services.	90
String	**name** The name of the declared reference.	89
String	**target** The target property of the satisfied reference.	89

Constructor Summary	Page
SatisfiedReferenceDTO()	90

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

name

```
public String name
```

The name of the declared reference.

This is declared in the `name` attribute of the `reference` element of the component description.

See Also:
ReferenceDTO.name

---

target

```
public String target
```

The target property of the satisfied reference.

This is the value of the component property whose name is the concatenation of the declared reference name and ".target". This must be null if no target property is set for the reference.

boundServices

```
public org.osgi.framework.dto.ServiceReferenceDTO[] boundServices
```

The bound services.

Each org.osgi.framework.dto.ServiceReferenceDTO in the array represents a service bound to the satisfied reference. The array must be empty if there are no bound services.

Constructor Detail

SatisfiedReferenceDTO

```
public SatisfiedReferenceDTO()
```

Class UnsatisfiedReferenceDTO

org.osgi.service.component.runtime.dto

```
java.lang.Object
  └org.osgi.dto.DTO
      └org.osgi.service.component.runtime.dto.UnsatisfiedReferenceDTO
```

```
public class UnsatisfiedReferenceDTO
extends org.osgi.dto.DTO
```

A representation of an unsatisfied reference.

Since:
1.3
NotThreadSafe

Field Summary		Page
String	**name** The name of the declared reference.	91
String	**target** The target property of the unsatisfied reference.	91
org.osgi.framework.dto.ServiceReferenceDTO[]	**targetServices** The target services.	92

Constructor Summary	Page
UnsatisfiedReferenceDTO()	92

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

name

```
public String name
```

The name of the declared reference.

This is declared in the `name` attribute of the `reference` element of the component description.

See Also:
ReferenceDTO.name

---

target

```
public String target
```

The target property of the unsatisfied reference.

This is the value of the component property whose name is the concatenation of the declared reference name and ".target". This must be null if no target property is set for the reference.

targetServices

```
public org.osgi.framework.dto.ServiceReferenceDTO[] targetServices
```

The target services.

Each org.osgi.framework.dto.ServiceReferenceDTO in the array represents a target service for the reference. The array must be empty if there are no target services. The upper bound on the number of target services in the array is the upper bound on the cardinality of the reference.

Constructor Detail

UnsatisfiedReferenceDTO

```
public UnsatisfiedReferenceDTO()
```


8 Considered Alternatives

8.1 Field injection of component activation objects

Activation fields are only set at component activation. They cannot be modified after that for the modified or deactivate life cycle events because of atomicity issues. The following was deleted:

- 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.

8.2 Component Reclamation

We agreed that the SCR implementation should provide some delay in reclaiming singleton and bundle scope services rather than create new markup for this. Also during CPEG discussion, we concluded that this could only apply to singleton and bundle scope services anyway since prototype services must be reclaimed when released. The following was deleted:

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.

8.3 Partitioned Map Field Type

Issues with concurrency and field-option updated led to the removal of this support. Using a custom list or collection of map.entry with field-option=update can be used to provide the desired behavior.

A new field type is introduced called a partitioned map. The type of this field must be one of

- Map<K,V>
- A subtype of Map<K,V>

V can be S or List<S> where S is one of the types supported by the field-collection-type attribute. The former is a single-valued map while the latter is a multi-valued map or multimap. If a subtype of Map is used, a subtype of List can also be used for a multimap and the policy must be dynamic and the field-option must be update.

Partitioned maps must use multiple cardinality like Collection and List field types. @Reference defaults are the same as Collection and List.

When using a partitioned map, the partition-key attribute (@Reference.partitionKey) must be specified. The value of this attribute is the name of the service property which will be used to partition the bound services into the map. If a target service does not specify a service property with the partition-key name or if the value of that service property cannot be coerced to the type K, then the target service will not be bound to this reference.

For a single-valued map, for each unique value of the service property, the highest ranked target service is bound and put in the map under the key of the service property value coerced to type K. For a multimap, for each unique value of the service property, all the target services are bound and put into the map as a List, sorted using the same ordering as ServiceReference.compareTo based upon service ranking and service id, under the key of the service property value coerced to type K.

If the service property has multiple values, that is, the value is an array or collection, then, for each unique value, the service must be placed into the partitions for which a value can be coerced to type K.

If field-option=update is used with a multimap, then SCR does not manage the List in the multimap and will not provide any sorting for the list.

8.4 Converter

The specification will not require or refer to the Converter specification. SCR implementations can use Converter as an implementation detail if they wish.

The DS specification is to be updated to replace the rules for component property mapping and coercing property values to use the new Converter package from RFC 215 instead. Care must be taken to ensure backward compatibility is preserved with DS 1.3 spec.

Beyond the standard conversions supplied by the Converter package, SCR will need to add an additional conversion rule.

SCR must add a rule to convert from a String to a Class using the component's bundle to load the class named by the String using Bundle.loadClass(String).

9 Security Considerations

This design introduces no new security considerations.

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