RFC 190 - Declarative Services Enhancements

Draft

71 Pages

Abstract

Declarative Services provide nice functionality to implement Dependency Injection programming in OSGi based applications. One of the goals is to limit the requirement to use OSGi specific API. This RFC proposes extensions towards this goal. In addition Declarative Services currently lacks a proper diagnostic API to introspect components.

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

0.3 Feedback

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

0.4 Table of Contents

0.5 Terminology and Document Conventions

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

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

0.6 Revision History

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

Revision	Date	Comments
Initial	Sept. 17 2012	Initial version from RFP Felix Meschberger, Adobe Systems Incorporated, fmeschbe@adobe.com
Update	Sept. 24 2012	Updates from Basel F2F: • Integrate Administrative API and design it to be DTO-style • Simplify security (ServicePermission [ServiceComponentRuntime, GET] is enough) Felix Meschberger, Adobe Systems Incorporated, fmeschbe@adobe.com

Revision	Date	Comments
Update	06/06/13	Update from Orlando F2F and BJ's feedback on the CPEG mailing list • Relable the administrative API as the diagnostic API • Fleshed out annotation inheritance but suggest to actually remove it (section 5.7.4, Supporting Inheritance) • Added section 5.9, Service Scopes Felix Meschberger, Adobe Systems Incorporated, fmeschbe@adobe.com
Update	07/08/13	Update from Palo Alto F2F • Removed separate service annotations • Removed annotation inheritance • Removed setting properties through the component • Updated DTOs • New suggestion for property annotation Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	07/19/13	Update from CPEG Call (18/07/13) • Removed alternative property annotation proposals • Clarified annotation based approach Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	09/18/13	Update after BJs review, major changes: • Removed props attribute from @Component annotation and allow multiple annotation arguments in the lifecycle methods • Remove DISPOSED state Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	01/10/13	Update with feedback from CPEG call: • Add information (back) from annotations as properties to descriptor XML Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	17/10/13	Update from Bug 2515 Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	24/10/13	Update from CPEG call • Clarified annotation field to property mapping

1 Introduction

This Declarative Services Enhancements RFC defines functionality currently implemented in some implementations of the specification or currently requiring special component code as part of the OSGi Declarative Services Specification.

2 Application Domain

Declarative Services (chapter 121 in the OSGi specifications) defines a POJO programming model for OSGi services. This model requires Service Component class be implemented in a certain way and the XML component descriptions be authored.

2.1 Terminology + Abbreviations

DS Declarative Services

POJO Plain old Java Object; term use for objects not implementing and framework specific plumbing such as Servlet API, Spring API, or OSGi API.

SCR Service Components Runtime; generally the implementation of the Declarative Services Specification; also the name of the Apache Felix implementation (Apache Felix SCR).

3 Problem Description

3.1 Management

There is no official API yet to introspect and thus manage the declared service components. To work around this missing functionality the Apache Felix project defined such an API which is also implemented by current versions of the Eclipse Equinox implementation.

This current API has some short-comings which are addressed by a new proposal.

3.2 Bound Service Properties

As of DS Version 1.1 the service registration properties of bound services can be provided to the components using an optional `java.lang.Map` argument. While this allows for great capability introspecting the bound service it lacks support for ordering defined for `org.osgi.framework.ServiceReference`.

The solution applied today is to either use the greedy service binding policy option as defined in DS Version 1.2 or to implement such ordering in the component itself. Such implementation, though, is pure template code and thus error prone load to developers.

4 Requirements

R-1 The solution MUST define a diagnosis API to introspect declared components.

R-2 The solution MUST make it possible to leverage the ordering capability of the `ServiceReference` along with the service instance provisioning through the event method by allowing the new signature:

```
void <method-name>(<parameter-type>, ServiceReference);
```

R-3 The solution MUST define a new way to define component properties by referencing a java annotation class. The fields of this annotation class are used to define properties for the component.

R-4 The solution MUST define the `osgi.extender` capability for DS in accordance with the core specification rules for the `osgi.extender` name space.

R-5 The solution MUST support targeted PIDs according to Configuration Admin 1.5.

5 Technical Solution

5.1 Diagnostic API

The diagnostic API is structured after the component descriptor within its own package `org.osgi.service.component.runtime.` The `ServiceComponentRuntime` service interface is the API entry point. `It` is registered by the DS implementation and provides access to properties of the implementation and to the properly declared components. Any components whose descriptor cannot be validated is considered unknown and thus is not available through the `ServiceComponentRuntime` service.

Each component declaration is accessible through the `ServiceComponentRuntime` as an instance of the `ComponentDescription` class. The `ComponentDescription` provides access to the static declaration.

Components actually are available from the `ServiceComponentRuntime` as `ComponentConfiguration` instances. Each `ComponentConfiguration` links back to its declaring `ComponentDescription`.

Since a single declaration may be activated multiple times – for example due to multiple factory configurations – a single `ComponentDescription` instance may be referred to by multiple `ComponentConfiguration` instances.

To cover the same difference between the declaration of references and actually bound references, the `ComponentDescription` object provides the declared references as `Reference` objects while the `ComponentConfiguration` returns `BoundReference` object representing actually bound services.

To simplify remote management the `ComponentDescription`, `ComponentConfiguration`, `Reference`, and `BoundReference` types are defined as DTO-style classes and integrate with the API defined by RFC-185, Data Transfer Objects [3]..

A bundle wishing to access the DTOs must have ServicePermission[ServiceComponentRuntime, GET] to get the ServiceComponentRuntime service.

5.2 Event Method Signature

A new supported signature for event strategy methods is added to the end of the list of supported signatures in Section 112.3.2, Event Methods:

```
void <method-name>(<parameter-type>, ServiceReference);
```

This signature is only supported if the component is declared in a descriptor with namespace `http://www.osgi.org/xmlns/scr/v1.3.0` or newer.

5.3 API version

The DS API is exported as version 1.3 to reflect these updates.

5.4 XML Descriptor Namespace

The XML descriptor namespace is changed to

`http://www.osgi.org/xmlns/scr/v1.3.0`

New functionality defined in this specification requires component to be registered with this namespace. Otherwise, for backwards compatibility reasons, neither the added event method signature nor the new prototype service scope an be used.

5.5 Extender Capability

The DS implementation bundle must declare the following extender capability:

```
Provide-Capability: osgi.extender;
       osgi.extender="osgi.component";
       uses:="org.osgi.service.component,org.osgi.service.component.runtime";
       version:Version="1.3"
```

5.6 Extension to Annotation Support

5.6.1 Custom annotations to define properties

Component properties can be defined through custom annotation classes containing the property names together with their default values:

```
@interface Config {
    boolean enabled() default true;
    String[] names() default {"a", "b"};
    String topic() default MyComponent.DEFAULT_TOPIC_PREFIX + "/topic";
}
@Component
public class MyComponent {

    static final String DEFAULT_TOPIC_PREFIX = "topic.prefix";

    protected void activate(Config configuration) {
        String t = configuration.topic();

    }
}
```

The lifecycle methods for activation, deactivation and modification can use a signature which allows to specify one or more annotation classes as arguments. The fields of all annotations from the lifecycle methods are added as properties to the component xml. The order of processing is: first annotations from the activate method, followed by annotations from the modified method and finally those from the deactivate method. If such a lifecycle method has more than one annotation these are processed in the order of the method arguments.

New signatures for the lifecycle methods are supported, taking one or more arguments of annotation classes. These are additional possible arguments. The following details the search for a lifecycle method:

1. The method takes a single argument and the type of the argument is org.osgi.service.component.ComponentContext.
2. The method takes a single argument and the type of the argument is org.osgi.framework.BundleContext.
3. The method takes one or more arguments and all arguments are annotation types.
4. The method takes a single argument and the type of the argument is the java.util.Map.
5. For deactivation methods only: The method takes a single argument and the type of the argument is the int.
6. For deactivation methods only: The method takes a single argument and the type of the argument is the java.lang.Integer.

7. The method takes two or more arguments and the type of each argument must be org.osgi.service.component.ComponentContext, org.osgi.framework.BundleContext, java.util.Map or an annotation type. For the deactivation method int or java.lang.Integer are allowed types as well. If multiple methods match this rule, this implies the method name is overloaded and SCR may choose any of the methods to call.

8. The method takes zero arguments.

5.6.2 Adding annotation fields to the component descriptor

Each field of an annotation with a default value is mapped to a component property. The name of the field is converted to the property name as follows: Each character and number are used as is, a single underscore is converted into a dot and two consecutive underscores are mapped to a single underscore. The processing starts at the beginning of the name. Examples:

Field Name	Property Name
myProperty143	myProperty143
some_prop	some.prop
another__prop	another_prop
three___prop	three_.prop

This processing means that there is currently no way to define a mapping for a property containing the character sequence of a dot followed by an underscore.

The type can directly be derived from the type of the field. All scalar types and the String type can be directly used. If the type is Class or Class[], the corresponding property gets the type String or String[]. If the type is an enumeration type or an array thereof, the corresponding type is String or String[] as well. ~~All types are supported, except Class, enum or~~ Fields with an annotation type or array~~s of these three types~~thereof are not supported. At runtime a field using one of these types always returns the default value, if the field doesn't have a default value null is returned. A tool processing the annotations should ~~-~~throw an error during the processing in this case. It's in general discouraged to us such types.

The default value of the annotation field is used as the value of the component property in the descriptor, for values of type Class Class.getName() is used, for enumeration values the name() method is used. ~~-~~ A field without a default value is not added to the component descriptor.

5.6.3 Mapping component properties to annotation fields

~~If the field does not have a default value, for numbers and char 0 is used as the default value, for a string the empty string. If the field has an array type, the same non-array default value is used for the property.~~

If an annotation is used within a lifecycle method, DS creates an implementation conforming to this annotation and maps the available component properties to the fields. If there is no property for a field, the default value of the annotation field is returned. If the field does not have a default value, for numbers and char 0 is used as the value, everything else gets the value null including arrays of any type. ~~As all annotation fields are written out as component properties, a value for each field exists for the component and no additional default mechanism is needed at runtime:~~The mapping from component properties to fields is done as follows:

- the name of the field is mapped to the property name as described above
- The property value might need to be mapped to the value of the annotation field:
  - as configurations might contain any object type a component property might have any type as well. DS only supports scalar types, Strings, arrays, lists or vectors thereof. Lists and vectors are treated

like arrays (see below). If an annotation field is accessed which maps to any other type a ComponentException is thrown.

- if cardinality and type are the same, the property value is used.
- if the type is the same, but the cardinality is different, a single property value is mapped to an array with exactly this value. If the property value is an array but the annotation field expects a single value, the first value of the array is returned.
- If the type differs, the property value is converted to a string using toString() on the value and then passed into the valueOf(String) method of the type class. For arrays this is done for each value. If the cardinality differs the same rules apply as explained in the previous point.
- If the type of the annotation field is Class, the property value is converted to a string using toString() and the class is tried to be load from the component's bundle classloader. For arrays this is done for each value. If a class is not found using the component's bundle classloader a ComponentException is thrown with the ClassNotFoundException as the cause.
- If the type of the annotation field is an enumeration, the property value is converted to a string using toString() and this string is passed into the valueOf() method of the enumeration. For arrays this is done for each value. If trying to instantiate the enum value throws an IllegalArgumentException this exception is caught and a ComponentException is thrown with the IllegalArgumentException as the cause.
- If an exception occurs during the conversion like a text is tried to be converted into a number, a ComponentException is thrown when the field of the annotation is accessed. The cause of the ComponentException is set to the origination exception. This allows component implementations to catch such a situation and act accordingly.

- Additional properties not defined in the annotation are ignored – if the implementation needs these additional properties, it can use a method signature which includes the properties map in addition to the annotation.

This new signature with annotations is only supported if the component is declared in a descriptor with namespace `http://www.osgi.org/xmlns/scr/v1.3.0` or newer.

5.6.4 Clarification of @Component annotation

The @Component annotation contains two different approaches to define properties through two different attributes: property and properties. In addition annotations can be used to define component properties as well. As annotation properties do not have an order when they are processed, the current specification does not define the order of the corresponding elements in the XML descriptor. However the order in the XML defines the order of processing and later properties with the same property name override previously declared ones.

This should be clarified by defining an order of processing:

1. properties defined through annotation classes used in the signature of the lifecycle methods.
    1. Annotations from the activate method in the order of parameters in the signature
    2. Annotations from the modified method in the order of parameters in the signature
    3. Annotations from the deactivate method in the order of parameters in the signature

2. property attribute from @Component
3. properties attribute from @Component
4. ~~property attribute from @Component~~

This means that the properties defined through annotations are added first to the XML descriptor, followed by all properties defined through the properties directive and finally the property statements are added.

5.7 Clarification of Section 112.7.1

As discussed in Bug 2515 section 112.7.1 needs some clarification. The proposed new text is:

SCR must track changes in the Configuration objects used in the component properties of a component configuration. Depending on the configuration policy and whether the modified attribute is set in the component description, a deletion of a configuration has a different outcome. If a configuration has set the policy-option to IGNORE, any configuration change has no influence on the component. For all other components the change is described in the below table.

A configuration change does not cause a deactivation and an attempt to reactivate a component if:

- the component description has the modified attribute set
- for every static reluctant reference, every bound service matches the new filter
- for every static greedy reference, the set of bound services would be the same for the new and the old filter
- for every dynamic reference with minimum cardinality of 1, there is at least one service satisfying the new filter

In all other cases, a configuration change will result in deactivating and attempting to reactive the component.

Policy	Create	Update	Delete
IGNORE	no effect on the component	no effect on the component	no effect on the component
REQUIRE	create and activate new component	call modified method	deactivate the component
OPTIONAL Single Configuration	configure the existing single instance	call modified method	call modified method
OPTIONAL Factory Configuration first or single instance	configure the existing single instance	call modified method	call modified method
OPTIONAL Factory Configuration more than one	create and activate new component	call modified method	deactivate the component

instance			

If a component has registered a service, but is not activated, the service properties change based on the new configuration.

5.8 Integration with the Configuration Admin Service

DS integrates with the Configuration Admin Service. Therefore implementations of DS must support the latest additions to the Configuration Admin Service:

- Targeted PIDs have been introduced in Configuration Admin 1.5. Section 104.3.3, Extenders and Targeted PIDs, requires extenders such as DS to properly support Targeted PIDs.
- Configuration Admin 1.4 introduced *multi-location* binding. DS implementations must make sure these bindings are properly supported. If the multilocation is just the question mark, no additional checks must be made, as this configuration can be used by any bundle. If a region is specified, the DS implementation must check whether the component bundle has the required permission as outlined in Configuration Admin 1.4, Section 104.7.6.

5.9 Service Scopes

RFC 195, Service Scopes, defines a new mechanism to access services from the service registry. This mechanism allows to get new service instances on demand instead of either always the same instance globally (regular service) or per bundle (service factory).

RFC 195 specifies the changes to Declarative Services to cope with Service Scopes in section 5.3, Declarative Services:

- The `service.servicefactory` attribute is replaced by a new `service.scope` attribute defined in the DS descriptor.
- A new `reference.scope` attribute to define the service reference scope is defined in the DS descriptor.
- A new bind and unbind signature `void <method-name>(ServiceObjects);` is defined to support prototype scoped references.
- `@Component.servicefactory()` is deprecated in favor of the new `@Component.scope()` of type `ServiceScope`.
- `@Reference.scope()` of type `ReferenceScope` is added.

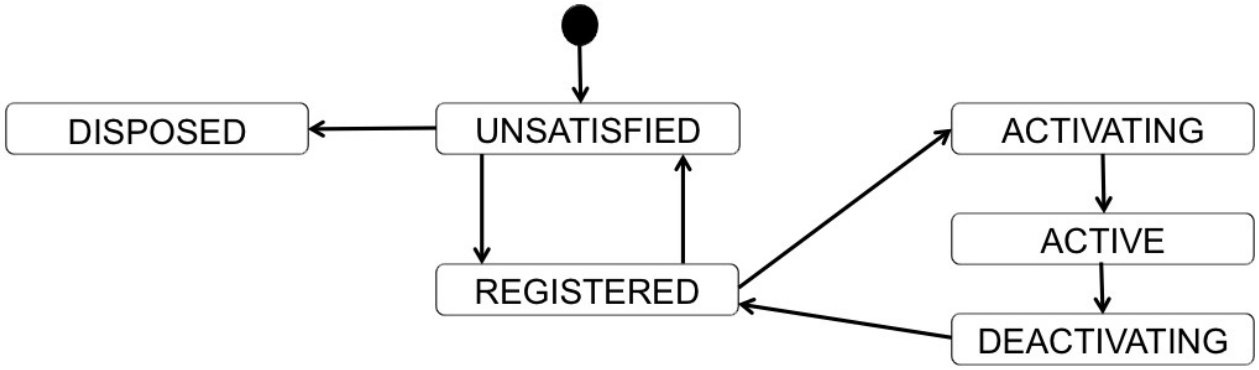
These changes specified in RFC 195, Service Scope, form an integral part of this RFC.

5.10 New Life Cycle States

Two additional states describing the life cycle of a component are added: activating and deactivating. A component is in the activating state when it is leaving the satisfied state while it is activated by calling the activate method of a component. This state is a transient state and an immediate component enters the active state once

it's activated and all other components enter the registered state. If activation fails, the component is back in the unsatisfied state.

If a component is deactivated, it enters the deactivating state during this process. Once deactivation is finished it is disposed.



6 Data Transfer Objects

The `ServiceComponentRuntime` service allows for the programmatic enablement and disablement of components as well as access to the state of components and component configurations. In particular the service provides these methods:

```
ComponentDescription getComponentDescription(Bundle, String)
Collection<ComponentDescription> getComponentDescriptions(Bundle...)
Collection<ComponentConfiguration>
getComponentConfigurations(ComponentDescription)
```

See the JavaDoc for details.

7 Java API

OSGi Javadoc

25.10.13 14:17

Package Summary		Page
org.osgi.service.component	Service Component Package Version 1.3.	16
org.osgi.service.component.annotations	Service Component Annotations Package Version 1.3.	28
org.osgi.service.component.runtime	Service Component Package Version 1.3.	53

Package org.osgi.service.component

```
@org.osgi.annotation.versioning.Version(value="1.3")
```

Service Component Package Version 1.3.

See:
[Description](#)

Interface Summary		Page
ComponentConstants	Defines standard names for Service Component constants.	*17*
ComponentContext	A Component Context object is used by a component instance to interact with its execution context including locating services by reference name.	*20*
ComponentFactory	When a component is declared with the `factory` attribute on its `component` element, the 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.	*26*
ComponentInstance	A ComponentInstance encapsulates a component instance of an activated component configuration.	*27*

Exception Summary		Page
ComponentException	Unchecked exception which may be thrown by the Service Component Runtime.	*24*

Package org.osgi.service.component Description

Service Component Package Version 1.3.

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="[1.3,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.component; version="[1.3,1.4)"
```

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_FACTORY** A service registration property for a Component Factory that contains the value of the `factory` attribute.	18
String	**COMPONENT_ID** A component property that contains the generated id for a component configuration.	18
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.	17
int	**DEACTIVATION_REASON_BUNDLE_STOPPED** The component configuration was deactivated because the bundle was stopped.	19
int	**DEACTIVATION_REASON_CONFIGURATION_DELETED** The component configuration was deactivated because its configuration was deleted.	19
int	**DEACTIVATION_REASON_CONFIGURATION_MODIFIED** The component configuration was deactivated because its configuration was changed.	19
int	**DEACTIVATION_REASON_DISABLED** The component configuration was deactivated because the component was disabled.	18
int	**DEACTIVATION_REASON_DISPOSED** The component configuration was deactivated because the component was disposed.	19
int	**DEACTIVATION_REASON_REFERENCE** The component configuration was deactivated because a reference became unsatisfied.	18
int	**DEACTIVATION_REASON_UNSPECIFIED** The reason the component configuration was deactivated is unspecified.	18
String	**REFERENCE_TARGET_SUFFIX** The suffix for reference target properties.	18
String	**SERVICE_COMPONENT** Manifest header specifying the XML documents within a bundle that contain the bundle's Service Component descriptions.	17

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

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 have an activate method. If a component instance has a suitable and accessible activate method, this method will be called when a component configuration is activated. If the activate method takes a `ComponentContext` argument, it will be passed the component instance's Component Context object. If the activate method takes a `BundleContext` argument, it will be passed the component instance's Bundle Context object. If the activate method takes a `Map` argument, it will be passed an unmodifiable Map containing the component properties.

A component instance may have a deactivate method. If a component instance has a suitable and accessible deactivate method, this method will be called when the component configuration is deactivated. If the deactivate method takes a `ComponentContext` argument, it will be passed the component instance's Component Context object. If the deactivate method takes a `BundleContext` argument, it will be passed the component instance's Bundle Context object. If the deactivate method takes a `Map` argument, it will be passed an unmodifiable Map containing the component properties. If the deactivate method takes an `int` or `Integer` argument, it will be passed the reason code for the component instance's deactivation.

ThreadSafe

Method Summary		Page
void	**disableComponent**(String name) Disables the specified component name.	23
void	**enableComponent**(String name) Enables the specified component name.	22
org.osgi.framework.BundleContext	**getBundleContext**() Returns the `BundleContext` of the bundle which contains this component.	22
ComponentInstance	**getComponentInstance**() Returns the Component Instance object for the component instance associated with this Component Context.	22
Dictionary<String,Object>	**getProperties**() Returns the component properties for this Component Context.	21
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.	23
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.	22
Object	**locateService**(String name) Returns the service object for the specified reference name.	21
Object	**locateService**(String name, org.osgi.framework.ServiceReference<?> reference) Returns the service object for the specified reference name and `ServiceReference`.	21
Object[]	**locateServices**(String name) Returns the service objects for the specified reference name.	21

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

```
Object 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 the Service Component Runtime catches an exception while activating the bound service.

---

locateService

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

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

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 the 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 the Service Component Runtime catches an exception while activating a bound service.

---

getBundleContext

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

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

Returns:
The `BundleContext` of the bundle containing 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:

1. The component instance is not a service, then no bundle can be using it as a service.
2. 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.
3. 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` **`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.

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.

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 the Service Component Runtime.

Constructor Summary	Page
`ComponentException(String message)` Construct a new ComponentException with the specified message.	24
`ComponentException(String message, Throwable cause)` Construct a new ComponentException with the specified message and cause.	24
`ComponentException(Throwable cause)` Construct a new ComponentException with the specified cause.	25

Method Summary		Page
Throwable	`getCause()` Returns the cause of this exception or `null` if no cause was set.	25
Throwable	`initCause(Throwable cause)` Initializes the cause of this exception to the specified value.	25

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

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

When a component is declared with the `factory` attribute on its `component` element, the 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	**newInstance**(Dictionary<String,?> properties) Create and activate a new component configuration.	26

Method Detail

newInstance

```
ComponentInstance 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 the Service Component Runtime is unable to activate the component configuration.

Interface ComponentInstance

org.osgi.service.component

```
@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.	27
Object	getInstance() Returns the component instance of the activated component configuration.	27

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

```
Object getInstance()
```

Returns the component instance of the activated component configuration.

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

Package org.osgi.service.component.annotations

```
@org.osgi.annotation.versioning.Version(value="1.3")
```

Service Component Annotations Package Version 1.3.

See:
Description

Enum Summary		Page
ConfigurationPolicy	Configuration Policy for the `Component` annotation.	35
ReferenceCardinality	Cardinality for the `Reference` annotation.	43
ReferencePolicy	Policy for the `Reference` annotation.	45
ReferencePolicyOption	Policy option for the `Reference` annotation.	47
ReferenceScope	Reference scope for the `Reference` annotation.	49
ServiceScope	Service scope for the `Component` annotation.	51

Annotation Types Summary		Page
Activate	Identify the annotated method as the `activate` method of a Service Component.	29
Component	Identify the annotated class as a Service Component.	30
Deactivate	Identify the annotated method as the `deactivate` method of a Service Component.	37
Modified	Identify the annotated method as the `modified` method of a Service Component.	38
Reference	Identify the annotated method as a `bind` method of a Service Component.	39

Package org.osgi.service.component.annotations Description

Service Component Annotations Package Version 1.3.

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)
public @interface Activate
```

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

The annotated method is the activate method of the Component.

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

Since:
1.1

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

Annotation Type Component

org.osgi.service.component.annotations

```
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.TYPE)
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 a Service Component Runtime implementation. 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."

Required Element Summary		Page
String	**configurationPid** The configuration PID for the configuration of this Component.	33
ConfigurationPolicy	**configurationPolicy** The configuration policy of this Component.	33
boolean	**enabled** Declares whether this Component is enabled when the bundle containing it is started.	32
String	**factory** The factory identifier of this Component.	31
boolean	**immediate** Declares whether this Component must be immediately activated upon becoming satisfied or whether activation should be delayed.	32
String	**name** The name of this Component.	30
String[]	**properties** Property entries for this Component.	32
String[]	**property** Properties for this Component.	32
ServiceScope	**scope** The service scope for the service of this Component.	34
Class<?>[]	**service** The types under which to register this Component as a service.	31
boolean	**servicefactory** **Deprecated.** *Since 1.3.*	31
String	**xmlns** The XML name space of the Component Description for this Component.	33

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

```
@Deprecated
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.

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.

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

---

enabled

```
public abstract boolean enabled
```

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

If `true`, this Component is enabled. If `false` or not specified, 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 `"key=value"`. The type of the property value can be specified in the key as `key: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 key, 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 OPTIONAL configuration policy is used.

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 PID for the configuration of this Component.

Allows the configuration PID for this Component to be different than the name of this Component.

If not specified, the name of this Component is used as the configuration PID 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.

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

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.	36
OPTIONAL Use the corresponding Configuration object if present but allow the component to be satisfied even if the corresponding Configuration object is not present.	35
REQUIRE There must be a corresponding Configuration object for the component configuration to become satisfied.	36

Method Summary		Page
String	**toString**()	36
static ConfigurationPolicy	**valueOf**(String name)	36
static ConfigurationPolicy[]	**values**()	36

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 a Service Component Runtime implementation. 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."

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 a Service Component Runtime implementation. 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)
public @interface Reference
```

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

The annotated method is a bind method of the Component.

This annotation is not processed at runtime by a Service Component Runtime implementation. 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
ReferenceCardinality	**cardinality** The cardinality of the reference.	40
String	**name** The name of this reference.	39
ReferencePolicy	**policy** The policy for the reference.	40
ReferencePolicyOption	**policyOption** The policy option for the reference.	41
ReferenceScope	**scope** The requested service scope for this Reference.	41
Class<?>	**service** The type of the service to bind to this reference.	40
String	**target** The target filter for the reference.	40
String	**unbind** The name of the unbind method which is associated with the annotated bind method.	40
String	**updated** The name of the updated method which is associated with the annotated bind method.	41

Element Detail

name

```
public abstract String name
```

The name of this reference.

If not specified, the name of this reference is based upon the name of the method being annotated. If the method name begins with `bind`, `set` or `add`, that is removed.

Default:
""

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

service

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

The type of the service to bind to this reference.

If not specified, the type of the service to bind is based upon the type of the first argument of the method being annotated.

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

cardinality

```
public abstract ReferenceCardinality cardinality
```

The cardinality of the reference.

If not specified, the reference has a `1..1` cardinality.

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

policy

```
public abstract ReferencePolicy policy
```

The policy for the reference.

If not specified, the `STATIC` reference policy is used.

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

target

```
public abstract String target
```

The target filter for the reference.

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

unbind

```
public abstract String unbind
```

The name of the unbind method which is associated with the annotated bind method.

To declare no unbind method, the value `"-"` must be used.

If not specified, the name of the unbind method is derived from the name of the annotated bind method. If the annotated method name begins with `bind`, `set` or `add`, that is replaced with `unbind`, `unset` or `remove`, respectively, to derive the unbind method name. Otherwise, `un` is prefixed to the annotated method name to derive the unbind method name. The unbind method is only set if the component type contains a method with the derived name.

Default:
""

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

---

policyOption

```
public abstract ReferencePolicyOption policyOption
```

The policy option for the 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."

---

updated

```
public abstract String updated
```

The name of the updated method which is associated with the annotated bind method.

To declare no updated method, the value `"-"` must be used.

If not specified, the name of the updated method is derived from the name of the annotated bind method. If the annotated method name begins with `bind`, `set` or `add`, that is replaced with `updated` to derive the updated method name. Otherwise, `updated` is prefixed to the annotated method name to derive the updated method name. The updated method is only set if the component type contains a method with the derived name.

Default:
""

Since:
1.2

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

---

scope

```
public abstract ReferenceScope scope
```

The requested service scope for this Reference.

If not specified, the `bundle` service scope is requested.

Default:
ReferenceScope.BUNDLE
Since:
1.3
See Also:
"The scope 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.	44
MANDATORY The reference is mandatory and unary.	43
MULTIPLE The reference is optional and multiple.	44
OPTIONAL The reference is optional and unary.	43

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

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.	45
STATIC The static policy is the most simple policy and is the default policy.	45

Method Summary		Page
String	**toString**()	46
static ReferencePolicy	**valueOf**(String name)	46
static ReferencePolicy[]	**values**()	46

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 the event strategy 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.	47
RELUCTANT The reluctant policy option is the default policy option for both `static` and `dynamic` reference policies.	47

Method Summary		Page
String	**toString**()	48
static ReferencePolicyOption	**valueOf**(String name)	48
static ReferencePolicyOption[]	**values**()	48

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.	49
PROTOTYPE If the referenced service has prototype service scope, then each instance of the component with this reference can receive a unique instance of the service.	49

Method Summary		Page
String	**toString**()	50
static ReferenceScope	**valueOf**(String name)	50
static ReferenceScope[]	**values**()	50

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 referenced service has prototype service scope, then each instance of the component with this reference can receive a unique instance of the service. If the referenced service does not have prototype service scope, then no service object will be received.

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 will be registered as a bundle scope service and an instance of the component will be created for each bundle using the service.	51
DEFAULT Default element value for annotation.	52
PROTOTYPE When the component is registered as a service, it will be registered as a prototype scope service.	52
SINGLETON When the component is registered as a service, it will be registered as a bundle scope service but only a single instance of the component will be used for all bundles using the service.	51

Method Summary		Page
String	**toString**()	52
static ServiceScope	**valueOf**(String name)	52
static ServiceScope[]	**values**()	52

Enum Constant Detail

SINGLETON

```
public static final ServiceScope SINGLETON
```

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

BUNDLE

```
public static final ServiceScope BUNDLE
```

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

PROTOTYPE

```
public static final ServiceScope PROTOTYPE
```

When the component is registered as a service, it will be registered as a prototype scope 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.runtime

```
@org.osgi.annotation.versioning.Version(value="1.3")
```

Service Component Package Version 1.3.

See:
Description

Interface Summary		Page
ServiceComponentRuntime	The `ServiceComponentRuntime` service represents the Declarative Services main controller also known as the Service Component Runtime or SCR for short.	*67*

Class Summary		Page
BoundReference	The `BoundReference` interface represents the actual service binding of a service reference declared in the reference element of the component declaration.	*54*
ComponentConfiguration	The `ComponentConfiguration` interface represents an actual instance of a declared `ComponentDescription`.	*56*
ComponentDescription	The `Component` interface represents the declaration of a component in a Declarative Services descriptor.	*60*
Reference	The `Reference` interface represents a single reference (or dependency) to a service used by a Component as declared in the `reference` elements of a Declarative Services descriptor.	*64*

Package org.osgi.service.component.runtime Description

Service Component Package Version 1.3.

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="[1.3,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.component; version="[1.3,1.4)"
```

Class BoundReference

org.osgi.service.component.runtime

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

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

The `BoundReference` interface represents the actual service binding of a service reference declared in the reference element of the component declaration.

Since:
1.3
Version:
$Id: 5d17c4cfdf9a25336ba9498daf0cb28ddded7c04 $
NotThreadSafe

Field Summary		Page
Reference	**reference** Returns the `component/reference` element of the component descriptor defining this bound reference.	54
boolean	**satisfied** Returns whether this reference is satisfied.	54
org.osgi.dto.framework.ServiceReferenceDTO[]	**serviceReferences** An array of `ServiceReferenceDTO` instances representing the bound services.	55
String	**target** The value of the actual target value used to select services to bind to.	55

Constructor Summary	Page
BoundReference()	55

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

Field Detail

reference

```
public Reference reference
```

Returns the `component/reference` element of the component descriptor defining this bound reference.

---

satisfied

```
public boolean satisfied
```

Returns whether this reference is satisfied. An optional reference is always satisfied. Otherwise `true` is only returned if at least one service is bound.

target

```
public String target
```

The value of the actual target value used to select services to bind to. Initially (without overwriting configuration) this method provides access to the `component/reference.target` attribute of the reference declaration. If configuration overwrites the target property, this method returns the value of the component property whose name is derived from the reference name plus the suffix *.target*. If no target property exists this field is set to `null`.

serviceReferences

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

An array of `ServiceReferenceDTO` instances representing the bound services. If no services are actually bound, this field is set to `null`.

Constructor Detail

BoundReference

```
public BoundReference()
```

Class ComponentConfiguration

org.osgi.service.component.runtime

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

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

The `ComponentConfiguration` interface represents an actual instance of a declared `ComponentDescription`. These instances are called *configurations* in the Declarative Services specification hence the name.

Since:
1.3
Version:
$Id: 9d2a68e7a6f4be379a58a9e0b5e82a5df9095a05 $
NotThreadSafe

Field Summary		Page
BoundReference[]	**boundReferences** An array of `BoundReference` instances representing the service references bound to this component configuration.	58
ComponentDescription	**component** The declaration of this component configuration.	58
String	**configurationPid** The `service.pid` property of the configuration properties provided by the Configuration Admin service for this component configuration or `null` if no configuration from the Configuration Admin is provided to this component configuration.	58
Map<String,Object>	**properties** A map of the actual properties provided to the component configuration.	58
int	**state** The current state of this component configuration, which is one of the `STATE_*` constants defined in this interface.	58
static int	**STATE_ACTIVATING** A registered component is being activated (value is 4).	57
static int	**STATE_ACTIVE** A component is in the active state (value is 8).	57
static int	**STATE_DEACTIVATING** The Component is being deactivated either because it is being disabled or because a dependency is not satisfied any more (value is 16).	58
static int	**STATE_REGISTERED** The state of a component when it becomes satisfied (value is 2).	57
static int	**STATE_UNSATISFIED** The initial state of a component (value is 1).	57

Constructor Summary	Page
ComponentConfiguration()	59

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

Field Detail

STATE_UNSATISFIED

```
public static final int STATE_UNSATISFIED = 1
```

The initial state of a component (value is 1).

When the component becomes satisfied it enters the activating state to be activated.

STATE_REGISTERED

```
public static final int STATE_REGISTERED = 2
```

The state of a component when it becomes satisfied (value is 2).

A component configuration becomes satisfied when the following conditions are all satisfied:

- The component is enabled.
- If the component description specifies configuration-policy=required, then a Configuration object for the component is present in the Configuration Admin service.
- Using the component properties of the component configuration, all the component's references are satisfied. A reference is satisfied when the reference specifies optional cardinality or there is at least one target service for the reference.

If the component provides a service, an entry is added to the service registry for the services the component provides. Once any of the listed conditions are no longer true, the component configuration becomes unsatisfied and the registrations are removed from the service registry.

Immediate components directly enter the activating state. Delayed and factory service components enter the activating state when the service is retrieved from the service factory for the first time.

If activation fails, the component remains in the registered state.

STATE_ACTIVATING

```
public static final int STATE_ACTIVATING = 4
```

A registered component is being activated (value is 4). Depending on the type of the component this may include the following steps:

- Create the component instance
- Bind available references
- Call the activator method (if any)

If activation succeeds the component enters the active state.

If activation fails the component falls back to the registered state.

STATE_ACTIVE

```
public static final int STATE_ACTIVE = 8
```

A component is in the active state (value is 8). The activate state means the following depending on the type of component:

1. The component is an immediate component
2. The component is a delayed or service factory component and at least one consumer has retrieved the provided service

3. The component is an instance of a Component Factory component created with the ComponentFactory.newInstance(java.util.Dictionary) method

If the component becomes unsatisfied it is being deactivated and enters the deactivating state.

If the component is a Component Factory created instance and is disposed off with the ComponentInstance.dispose() method it is being destroyed and also enters the deactivating state.

If the last consumer of a delayed or service factory component ungets the provided service, the component instance is destroyed and the component enters the registered state.

STATE_DEACTIVATING

```
public static final int STATE_DEACTIVATING = 16
```

The Component is being deactivated either because it is being disabled or because a dependency is not satisfied any more (value is 16). After deactivation the Component enters the unsatisfied state.

---

component

```
public ComponentDescription component
```

The declaration of this component configuration.

---

state

```
public int state
```

The current state of this component configuration, which is one of the STATE_* constants defined in this interface.

---

configurationPid

```
public String configurationPid
```

The service.pid property of the configuration properties provided by the Configuration Admin service for this component configuration or null if no configuration from the Configuration Admin is provided to this component configuration.

---

properties

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

A map of the actual properties provided to the component configuration. This map provides the same content as the ComponentContext.getProperties() method.

---

boundReferences

```
public BoundReference[] boundReferences
```

An array of BoundReference instances representing the service references bound to this component configuration. null is returned if the component configuration has no bound references.

Constructor Detail

ComponentConfiguration

```
public ComponentConfiguration()
```

Class ComponentDescription

org.osgi.service.component.runtime

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

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

The `Component` interface represents the declaration of a component in a Declarative Services descriptor.

Since:
1.3
Version:
$Id: dc999e7d8096c5dbfbc5a206c6a6c6c22a5fd00a $
NotThreadSafe

Field Summary		Page
String	**activate** The name of the method to be called when the component is being activated as defined in the `component.activate` attribute or `null` if not explicitly declared.	62
org.osgi.dto.framework.BundleDTO	**bundle** The bundle declaring this component.	61
String	**configurationPolicy** The configuration policy declared in the `component.configuration-policy` attribute.	63
String	**deactivate** The name of the method to be called when the component is being deactivated as defined in the `component.deactivate` attribute or `null` if not explicitly declared.	62
boolean	**defaultEnabled** Whether the component is declared to be enabled by default (`true`) as defined by the `component.enabled` attribute.	62
String	**factory** The component factory name from `component.factory` attribute or `null` if this component is not defined as a component factory.	61
long	**id** The unique ID of this component managed by the Service Component Runtime.	63
boolean	**immediate** Whether the component is an immediate or a delayed component as defined by the `component.immediate` attribute.	62
String	**implementationClass** The fully qualified name of the class implementing this component from the `component/implementation.class` attribute.	61
String	**modified** The name of the method to be called when the component's configuration is being updated as defined in the `component.modified` attribute or `null` if not declared.	63
String	**name** The name of the component defined in the `component.name` attribute which may be `null`.	61
Map<String,Object>	**properties** The declared properties of the component as defined by the `component/property` and `component/properties` elements.	62
Reference[]	**references** An array of `Reference` instances representing the service references (or dependencies) of this component as defined in the `component/reference` elements.	62

String	**scope** The service scope for the service of this Component as defined by the `service/scope` attribute.	*61*
String[]	**serviceInterfaces** An array of service names provided by this component or `null` if the component is not registered as a service as defined by the `component/service/provide.interface` attributes.	*62*

Constructor Summary	Page
ComponentDescription()	*63*

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

Field Detail

name

```
public String name
```

The name of the component defined in the `component.name` attribute which may be `null`.

---

bundle

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

The bundle declaring this component.

---

factory

```
public String factory
```

The component factory name from `component.factory` attribute or `null` if this component is not defined as a component factory.

---

scope

```
public String scope
```

The service scope for the service of this Component as defined by the `service/scope` attribute.

---

implementationClass

```
public String implementationClass
```

The fully qualified name of the class implementing this component from the `component/implementation.class` attribute.

---

defaultEnabled

```
public boolean defaultEnabled
```

Whether the component is declared to be enabled by default (`true`) as defined by the `component.enabled` attribute.

---

immediate

```
public boolean immediate
```

Whether the component is an immediate or a delayed component as defined by the `component.immediate` attribute.

---

serviceInterfaces

```
public String[] serviceInterfaces
```

An array of service names provided by this component or `null` if the component is not registered as a service as defined by the `component/service/provide.interface` attributes.

---

properties

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

The declared properties of the component as defined by the `component/property` and `component/properties` elements.

---

references

```
public Reference[] references
```

An array of `Reference` instances representing the service references (or dependencies) of this component as defined in the `component/reference` elements.

---

activate

```
public String activate
```

The name of the method to be called when the component is being activated as defined in the `component.activate` attribute or `null` if not explicitly declared.

---

deactivate

```
public String deactivate
```

The name of the method to be called when the component is being deactivated as defined in the `component.deactivate` attribute or `null` if not explicitly declared.

---

modified

```
public String modified
```

The name of the method to be called when the component's configuration is being updated as defined in the `component.modified` attribute or `null` if not declared.

---

configurationPolicy

```
public String configurationPolicy
```

The configuration policy declared in the `component.configuration-policy` attribute. If the component descriptor is a Declarative Services 1.0 descriptor or not configuration policy has been declared, the default value *optional* is returned.

The returned string is one of the three policies defined in the Declarative Services specification 1.1:

optional
: Configuration from the Configuration Admin service is supplied to the component if available. Otherwise the component is activated without Configuration Admin configuration. This is the default value reflecting the behaviour of Declarative Services 1.0

require
: Configuration is required. The component remains unsatisfied until configuartion is available from the Configuration Admin service.

ignore
: Configuration is ignored. No Configuration Admin service configuration is supplied to the component.

---

id

```
public long id
```

The unique ID of this component managed by the Service Component Runtime. This value is also available as the `component.id` service registration property of component configurations registered as services.

Constructor Detail

ComponentDescription

```
public ComponentDescription()
```

Class Reference

org.osgi.service.component.runtime

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

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

The `Reference` interface represents a single reference (or dependency) to a service used by a Component as declared in the `reference` elements of a Declarative Services descriptor.

Since:
1.3
Version:
$Id: 44060765b30df744f52a143ed719105722117a8a $
NotThreadSafe

Field Summary		Page
String	**bind** The name of the method called if a service is being bound to the component as defined by the `reference.bind` attribute or `null` if no such method is declared.	66
String	**interfaceName** The name of the service used by this reference as defined by the `reference.interface` attribute.	65
boolean	**multiple** Whether this reference is multiple as defined by the upper bound of the `reference.cardinality`.	65
String	**name** The name of this Reference as defined by the `reference.name` attribute or `null` if not declared.	65
boolean	**optional** Whether this reference is optional as defined by the lower bound of the `reference.cardinality`.	65
String	**policy** Whether the reference is statically or dynamically bound as defined by the `reference.policy` attribute.	65
String	**policyOption** Policy of handling of availability of a better service as defined by the `reference.policy-option` attribute.	65
ReferenceScope	**scope** The requested service scope for this Reference as defined by the `reference.scope` attribute.	66
String	**target** The value of the target property of this reference as defined by the `reference.target` attribute or `null` if not declared.	66
String	**unbind** The name of the method called if a service is being unbound from the component as defined by the `reference.unbind` attribute or `null` if no such method is declared.	66
String	**updated** The name of the method called if the bound service service is updated as defined by the `reference.updated` attribute or `null` if no such method is declared.	66

Constructor Summary	Page
Reference()	66

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

Field Detail

name

```
public String name
```

The name of this Reference as defined by the `reference.name` attribute or `null` if not declared.

---

interfaceName

```
public String interfaceName
```

The name of the service used by this reference as defined by the `reference.interface` attribute.

---

optional

```
public boolean optional
```

Whether this reference is optional as defined by the lower bound of the `reference.cardinality`. In other words this field is set to `true` if the cardinality is *0..1* or *0..n*.

---

multiple

```
public boolean multiple
```

Whether this reference is multiple as defined by the upper bound of the `reference.cardinality`. In other words this field is set to `true` if the cardinality is *0..n* or *1..n*.

---

policy

```
public String policy
```

Whether the reference is statically or dynamically bound as defined by the `reference.policy` attribute.

---

policyOption

```
public String policyOption
```

Policy of handling of availability of a better service as defined by the `reference.policy-option` attribute.

---

target

```
public String target
```

The value of the target property of this reference as defined by the `reference.target` attribute or `null` if not declared.

---

bind

```
public String bind
```

The name of the method called if a service is being bound to the component as defined by the `reference.bind` attribute or `null` if no such method is declared.

---

unbind

```
public String unbind
```

The name of the method called if a service is being unbound from the component as defined by the `reference.unbind` attribute or `null` if no such method is declared.

---

updated

```
public String updated
```

The name of the method called if the bound service service is updated as defined by the `reference.updated` attribute or `null` if no such method is declared.

---

scope

```
public ReferenceScope scope
```

The requested service scope for this Reference as defined by the `reference.scope` attribute.

Constructor Detail

Reference

```
public Reference()
```

Interface ServiceComponentRuntime

org.osgi.service.component.runtime

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

The ServiceComponentRuntime service represents the Declarative Services main controller also known as the Service Component Runtime or SCR for short. It provides access to the components managed by the Service Component Runtime.

This service differentiates between ComponentDescription and ComponentConfiguration. A ComponentDescription is the declaration of the component in the Declarative Services descriptor. A ComponentConfiguration is an actual instance of a declared ComponentDescription and is backed by an object instance of the implementation class name declared in the component.

Access to this service requires the ServicePermission[org.osgi.service.component.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
Version:
$Id: 0cf40154dcc16759a08f7f3391de7ad32c7df87f $
ThreadSafe

Method Summary		Page
void	**disableComponent**(ComponentDescription description) Disables this ComponentDescription if it is enabled.	69
void	**enableComponent**(ComponentDescription description) Enables this ComponentDescription if it is disabled.	69
Collection<ComponentConfiguration>	**getComponentConfigurations**(ComponentDescription description) Return a collection of component configurations created for the component description.	68
ComponentDescription	**getComponentDescription**(org.osgi.framework.Bundle bundle, String name) Return the ComponentDescription declared with the given name or null if no such component is declared by the given bundle or the bundle is not active.	68
Collection<ComponentDescription>	**getComponentDescriptions**(org.osgi.framework.Bundle... bundles) Returns the component descriptions declared by the given bundles or the component descriptions declared by all active bundles if bundles is null.	67
boolean	**isComponentEnabled**(ComponentDescription description) Whether this component is currently enabled (true) or not.	68

Method Detail

getComponentDescriptions

```
Collection<ComponentDescription> getComponentDescriptions(org.osgi.framework.Bundle... bundles
)
```

Returns the component descriptions declared by the given bundles or the component descriptions declared by all active bundles if bundles is null. If the bundles have no declared components or the bundles are not active an empty collection is returned.

Parameters:
bundles - The bundles whose declared components are to be returned or null if the declared components from all active bundles are to be returned.

Returns:
The declared component descriptions of the given (active) `bundles` or the declared component descriptions from all active bundles if `bundle` is `null`. An empty collection is returned if no components are declared by active bundles.

---

getComponentDescription

```
ComponentDescription getComponentDescription(org.osgi.framework.Bundle bundle,
                                             String name)
```

Return the `ComponentDescription` declared with the given `name` or `null` if no such component is declared by the given `bundle` or the bundle is not active.

Parameters:
`bundle` - The bundle declaring the requested component
`name` - The name of the `ComponentDescription` to return
Returns:
The named component or `null` if none of the active bundles declare a component with that name.
Throws:
`NullPointerException` - if `name` or `bundle` is `null`.

---

getComponentConfigurations

```
Collection<ComponentConfiguration> getComponentConfigurations(ComponentDescription description
)
```

Return a collection of component configurations created for the component description. If there are no component configurations currently created, the collection is empty. This collection of configurations represents a snapshot of the current state.

Parameters:
`description` - The component description.
Returns:
The component configurations created for the given component description. An empty collection is returned if there are non.
Throws:
`NullPointerException` - if `description` or `null`.

---

isComponentEnabled

```
boolean isComponentEnabled(ComponentDescription description)
```

Whether this component is currently enabled (`true`) or not.

Initially this follows the `Component.enabled` attribute of the declaration and can be changed by the `enableComponent(ComponentDescription)` method, `disableComponent(ComponentDescription)` method, `ComponentContext.disableComponent(String)` or `ComponentContext.enableComponent(String)`.

Parameters:
`description` - The component description.
Returns:
Whether this component is currently enabled or not.
Throws:
`NullPointerException` - if `description` or `null`.
See Also:
`ComponentDescription.defaultEnabled`

---

enableComponent

`void` **`enableComponent`**`(`ComponentDescription `description)`

Enables this ComponentDescription if it is disabled. If the ComponentDescription is not currently disabled this method has no effect.

Parameters:
`description` - The component description to enable.
Throws:
`NullPointerException` - if `description` or is `null`.

---

disableComponent

`void` **`disableComponent`**`(`ComponentDescription `description)`

Disables this ComponentDescription if it is enabled. If the ComponentDescription is currently disabled this method has no effect.

Parameters:
`description` - The component description to disable.
Throws:
`NullPointerException` - if `description` or is `null`.

---

8 Considered Alternatives

8.1 Diagnostic API

The proposed diagnostic API is an evolution of the API already supported by the Apache Felix and Eclipse DS implementations. That existing API has a number of weaknesses which are addressed in the proposed API:

- Lack of a security model: Except for the ServicePermission[GET] to access the ScrService service no security is defined at all
- The service is badly named “ScrService”
- There is no distinction between a Component as a declared entity and a Component Configuration as an actual instance of the declared entity. This makes the API look strange when dealing with multiple Component Configurations for the same Component.

Thus the proposed API is a complete reqwrite of the existing API to better match the actual situation of a service component runtime.

8.2 @Component annotation inheritance

The original Bug 2138 asked for full support for inheritance of annotations. Such inheritance is already supported by the DS annotations provided by the Apache Felix project.

Yet full support is problematic because annotations are evaluated at build time based on build time dependencies available. Later at run time the static declarations are used to define properties, bind references and expose services. If a base class is modified between some expectations of that base class may not be met.

Consider for example a component `Extender` extending from the `Base` component. The `Base` component has an optional reference to Service `S1` at build time. At deployment time a new version of the `Base` component is deployed which besides the optional reference now has a mandatory reference to a Service `S2`. The descriptor created for `Extender` does not have this mandatory reference and thus may cause unexpected runtime errors (probably `NullPointerException`).

Another problem with full inheritance support is that implementations have to be exported. For the extending classes to have access to the base classes, those must be available in the class space of the extending class. This requires components to be exported. But this violates a basic assumption of DS which deems it best practice to not expose implementation details through export.

For these two reasons it was decided at the Basel F2F to support inheritance for components within the same bundle.

As it is very hard for tooling – up to impossible – to decide whether a (parent) class is within the same bundle, it was decided at the Palo Alto F2F to drop inheritance completely.

8.3 Create separate Service annotation (Bug 2140)

The original bug 2140 asked for the creation of a separate service annotation. However, due to impedance mismatch on the default of @Component.service() and a new @Service annotation, it was decided after the F2F meeting in Palo Alto to drop this additional annotation.

8.4 Component provided service properties (Bug 2250)

Different ways of supporting changes of service properties through the component have been discussed like returning a map from the activation method and/or having a setter method on the component context. However this would create several problems like how to update the properties and when especially with factory components. Therefore it was decided at the Palo Alto F2F to drop this enhancement.

8.5 Create separate Property annotation (Bug 2141)

The original bug 2141 asked for a separate property annotation. After extensive considerations this suggestion has been withdrawn and replaced with the new annotation based approach to define properties.

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

9 Security Considerations

The diagnostic API has security implications in that it allows to introspect into component declarations and instances which are otherwise not accessible. In addition the API provides functionality to actually disable or enable components, albeit this is only temporary and reverted by a system or bundle restart.

Thus the complete API should only be available to management agents. Since this is a simple have-it-or-not situation, any bundle requiring access to the diagnostic API must have the ServicePermission[ServiceComponentRuntime, GET] permission.

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

[3]. RFC 185, Data Transfer Objects

10.2 Author's Address

Name	Felix Meschberger
Company	Adobe Systems Incorporated
Address	Barfüsserplatz 6, 4055 Basel, Switzerland
Voice	+41 61 226 55 49
e-mail	fmeschbe@adobe.com

Name	Carsten Ziegeler
Company	Adobe Systems Incorporated
Address	Barfüsserplatz 6, 4055 Basel, Switzerland
Voice	+41 61 226 55 0
e-mail	cziegele@adobe.com

10.3 Acronyms and Abbreviations

10.4 End of Document