RFC 190 - Declarative Services Enhancements

Draft

82 Pages

Abstract

Declarative Services provide functionality to implement Dependency Injection programming in OSGi based applications. This RFC proposes describes adding an additional management API, better support of property handling through annotation interfaces, new features introduced in Configuration Admin 1.4 and 1.5, service scopes, and defines an extender namespace.

0 Document Information

0.1 License

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

Revision	Date	Comments
2nd	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
3rd	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
4th	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
5th	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
6th	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
7th	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

Revision	Date	Comments
8th	17/10/13	Update from Bug 2515 Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
9th	24/10/13	Update from CPEG call • Clarified annotation field to property mapping
10th	2013-11-11	API/Javadoc improvements (up through section 5.6) BJ Hargrave, IBM
11th	2013-11-15	Rewrite the Configuration Admin support changes. Added support for multiple PIDs. BJ Hargrave, IBM
12th	2013-11-19	Improved the annotation element to property name conversion rules. Added '$' to annotation element to property name conversion rules. Some other text improvements to clarify meaning, Reference RFC 208 AttributeDefinition annotation for the property name for an annotation element. BJ Hargrave, IBM
13th	2013-11-20	Updated with feedback from Peter Kriens. The signatures for event methods are expanded to allow more flexibility. The changes for prototype services are expanded to include the text from RFC 195. BJ Hargrave, IBM
14th	2013-11-22	Added diagram for DTOs. Removed reference to id element of AttributeDefinition annotation. Defined special string for component name that can be used in configurationPid(). New coercion table from property value to annotation element value. BJ Hargrave, IBM
15th	2013-12-06	Reviewed changes in CPEG call and accepted them. Minor changes to coercion rules including when no property is present. BJ Hargrave, IBM
16th	2013-12-16	Allow single Map argument for event methods. Add support for lookup strategy references. Added minimum cardinality property for references. BJ Hargrave, IBM
17th	2013-12-19	Reviewed changes in CPEG call and accepted them. BJ Hargrave, IBM
18th	2014-01-10	Updated targeted PID support to state that creation and deletion of Configurations with matching targeted PIDs are treated as a configuration modification. BJ Hargrave, IBM

Revision	Date	Comments
19th	2014-01-24	The minimum cardinality property is to be supported for older component namespaces. Configuration property names starting with *<refname>*. are reserved for future use by the specification. The DTOs were reviewed to ensure there were no cycles. There were none, but there were duplicate instances of ReferenceDTO. BoundReferenceDTO was changed to use the reference name rather than a ReferenceDTO. BJ Hargrave, IBM
20th	2014-02-07	Updated figure in 5.1 to fix errors in cardinalities. Refactored dtos into a package name ending with dto in conformance with updated dto package naming rules. BJ Hargrave, IBM
21st	2014-04-22	Bug 2668: Address service types of Map, ServiceReference and ServiceObjects in selecting the event method signature. BJ Hargrave, IBM
22nd	2014-06-03	Bug 2679: Side effects of enable/disable component are asynchronous. BJ Hargrave, IBM

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.

A number of changes have occurred over the past few years that affect DS:

- Apache Felix has implemented this specification and added a very popular management API. This same API was subsequently also used by Eclipse. The API is registered as an Admin service and provides methods to query the state of the components.
- Configuration Admin introduced a number of changes affecting DS.
  - *Targeted PIDs* — Targeted PIDs were introduced in Configuration Admin 1.5. These are PIDs that can distinguish between bundles and therefore target the configuration for a specific bundle. (See 104.3.2). Targeted PIDs allows for the PID to be further constrained by the bundle symbolic name, bundle version and bundle location. For ManagedService and ManagedServiceFactory services, Configuration Admin will always supply the configuration with the most targeted PID. Section 104.3.3, Extenders and Targeted PIDs, requires extenders such as SCR to properly support Targeted PIDs.
  - *Multiple PIDs* — Multiple PID support was introduced in Configuration Admin 1.4. Multiple PID support allows a configuration target to receive configurations for multiple PIDs. This is of more value with multi-location binding since a common configuration can be shared among a set of bundles.
  - *Multi-location binding* — Multi-location binding was introduced in Configuration Admin 1.4. Multi-location binding allows a configuration to avoid binding to a specific bundle location by prefixing the location with “?”. This way multiple bundles can share the same configuration. Java permissions can be used to limit a bundle's access to a configuration.
- 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).

DS supports Reference annotations that can be applied to *bind* methods. The reference name and service type can be inferred from the annotated method signature. A reference element in the component description is generated which references the annotated bind method.

DS supports four reference cardinality modes. In addition to supporting more than one reference, a reference can be optional or mandatory. That is, a reference can be satisfied with zero or one bound service.

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 the natural ordering defined for `org.osgi.framework.ServiceReference` which implements `Comparable`.

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.

3.3 Lookup Strategy

Reference annotations can only be applied to methods and so only really support the event strategy of referenced service access. To use only lookup strategy for a reference, you either cannot use DS annotations or you must supply an otherwise unused bind method to be annotated by the Reference annotation.

3.4 Deployer wants a mandatory reference

A developer can author a component to allow for an optional reference. The component can also be authored to support more than one reference. That is, the lower bound is 0 and the upper bound is 1 or n. However, a deployer may want to narrow the cardinality of the reference to require more than 0 bound services.

4 Requirements

R-1 The solution MUST define an API to introspect declared components and their configurations.

R-2 The solution MUST make it possible to leverage the natural ordering capability of `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.

R-6 The solution MUST support multi-location binding from Configuration Admin 1.4

R-7 The solution MUST support the multiple PIDs feature from Configuration Admin 1.4

R-8 The solution MUST support the Service scopes introduced in the core.

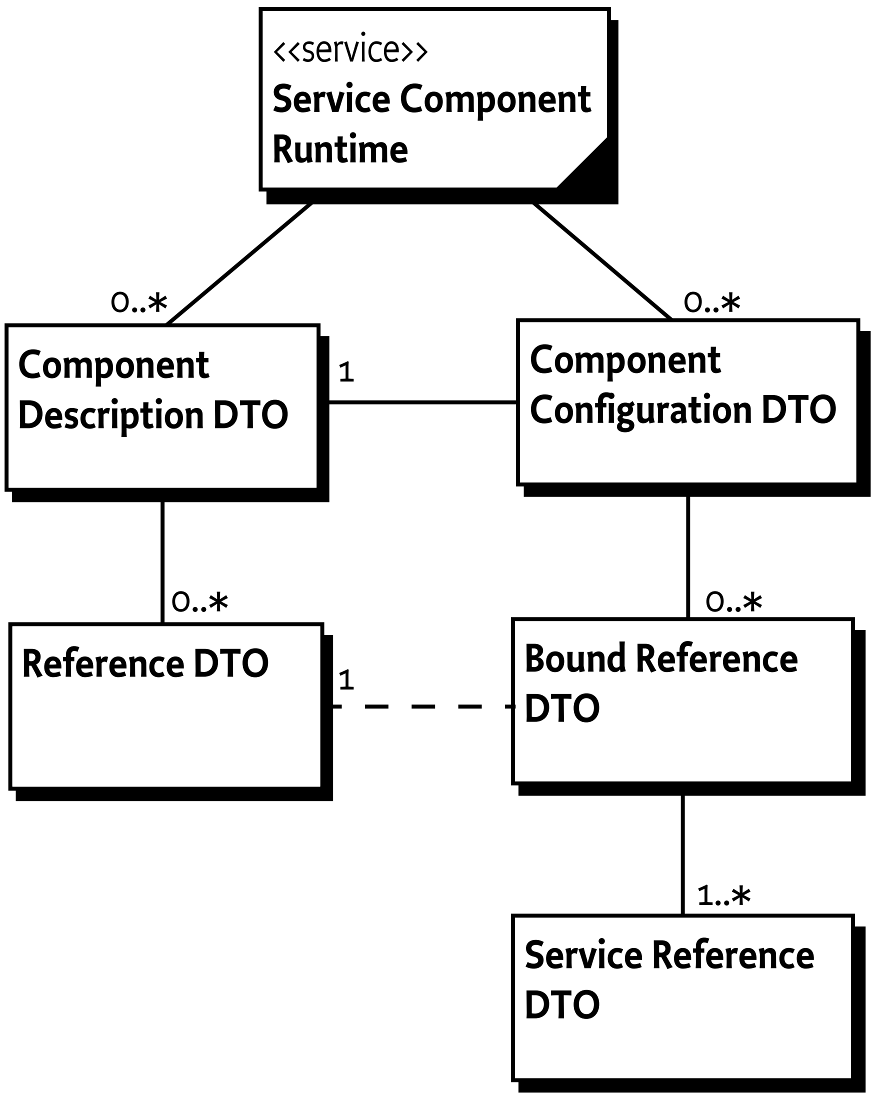
R-9 The solution MUST allow references to be declared in annotations without requiring a bind method.

R-10 The solution MUST allow a deployer to raise the minimum number of bound services for a reference to be considered satisfied.

5 Technical Solution

5.1 Introspective API

The introspective 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 SCR 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 `ComponentDescriptionDTO` class. The `ComponentDescriptionDTO` provides access to the static declaration.

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

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

To cover the same difference between the declaration of references and actually bound references, the `ComponentDescriptionDTO` object provides the declared references as `ReferenceDTO` objects while the `ComponentConfigurationDTO` returns `BoundReferenceDTO` object representing actually bound services.

To simplify remote management, the `ComponentDescriptionDTO`, `ComponentConfigurationDTO`, `ReferenceDTO`, and `BoundReferenceDTO` 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

The signatures for event strategy methods in Section 112.3.2, Event Methods are modified to allow for an new parameter, ServiceObjects and also to allow for additional flexibility in the ordering and number of parameters. Section 112.3.2 is updated to state:

The prototype of the events methods is

```
void <method-name>(<arguments>);
```

The event methods can take one or more arguments. Each argument must be of one of the following types:

- <service-type> – This is the bound service object.
- ServiceReference – This is the ServiceReference for the bound service.
- ServiceObjects – This is the ServiceObjects for the bound service. This argument type can only be used when the reference is `scope=prototype`.
- Map – This is a map of the service properties for the bound service.

A suitable method is selected using the following priority:

1. The method takes a single argument and the type of the argument is org.osgi.framework.ServiceReference. This method will receive the ServiceReference for the bound service.
2. The method takes a single argument and the type of the argument is org.osgi.framework.ServiceObjects. This signature can only be used when the reference is `scope=prototype`. This method will receive the ServiceObjects for the bound service.
3. The method takes a single argument and the type of the argument is the type specified by the reference's interface attribute. This method will receive the bound service.
4. The method takes a single argument and the type of the argument is assignable from the type specified by the reference's interface attribute. If multiple methods match this rule, this implies the method name is overloaded and SCR may choose any of the methods to call. This method will receive the bound service.
5. The method takes a single argument and the type of the argument is java.util.Map. This method will receive the service properties for the bound service.
6. The method takes two or more arguments and the type of each argument must be the type specified by the reference's interface attribute, assignable from the type specified by the reference's interface attribute, org.osgi.framework.ServiceReference, org.osgi.framework.ServiceObjects or java.util.Map. If multiple methods match this rule, this implies the method name is overloaded and SCR may choose any of the methods to call. In the case where the type of the bound service is Map, ServiceReference or ServiceObjects, the first argument of that type will receive the bound service. If the selected method has

more than one argument of that type, the remaining arguments of that type will receive the service properties Map, ServiceReference or ServiceObjects, respectively, for the bound service.

The above changes allow for more flexibility in the signatures of event methods, similar to the lifecycle methods, including multiple arguments of different type in any order.

The additional signatures are 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";
       version:Version="1.3"
```

5.6 Configuration Annotation Types

5.6.1 Custom annotations to define properties

Component properties can be defined through custom annotation types containing the property names, property types and default values. For example:

```
@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 them to specify one or more annotation types as arguments. The annotation elements with default values from all the annotation types used in the lifecycle method arguments are added via property elements to the generated component descriptions. The order of processing is: first annotation types used as arguments to the activate method, followed by annotation types used as arguments to from the modified method and finally annotation types used as arguments to the deactivate method. If a lifecycle method has more than one annotation type arguments, 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 types. 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 a single argument and the type of the argument is an annotation type.

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 Mapping Annotation Elements to Component Properties

Each element of an annotation type used as a lifecycle method argument is mapped to a component property. The property name is derived from the element name. Certain common property name characters, such as '.' are not not valid in Java identifiers. So the name of an annotation element is converted to the property name as follows:

- A single dollar sign ('$') is removed unless it is followed by another dollar sign in which case the two consecutive dollar signs ('$$') are converted to a single dollar sign.
- A single underscore ('_') is converted into a dot ('.') unless is it followed by another underscore in which case the two consecutive underscores ('__') are converted to a single underscore.

- All other characters and numbers are unchanged.

The processing starts at the beginning of the name. Examples:

Annotation Element Name	Component Property Name
myProperty143	myProperty143
$new	new
my$$prop	my$prop

Annotation Element Name	Component Property Name
_secret	.secret
another__prop	another_prop
three___prop	three_.prop
four_$__prop	four._prop
five_$_prop	five..prop

The property type can directly be derived from the type of the annotation element. All types supported for annotation elements can be used except for annotation types.

If the annotation element type is Class or Class[], the corresponding property type is String or String[] whose value comes from the `Class.getName()` method.

If the annotation element type is an enumeration type or an array thereof, the corresponding property type is String or String[] whose value comes from the `Enum.name()` method.

Annotation elements with an annotation type or array thereof are not supported. A tool processing the DS annotations should declare an error during processing in this case. Any default value is ignored.

If an annotation element has a default value, the tool processing the DS annotation must add a property element to the component description with the property name converted from the annotation element name and the property value from the annotation element's default value. Default values of type Class or an enumeration type are mapped to String values in the component description using `Class.getName()` and `Enum.name()` respectively. If an annotation element does not have a default value, the tool processing the DS annotation must not add a property element to the component description for the converted property name.

5.6.3 Mapping Component Properties to Annotation Elements

If an annotation type is used as an argument for a lifecycle method, SCR must create an object implementing the annotation type and maps the available component properties to the annotation elements. The name of the annotation element is converted to the property name as described above. The property value might need to be coerced to the type of the annotation element. In the following coercion table, the columns are source types (for example, component property type) and the rows are target types (for example, annotation element type). The property value is v; *number* is the primitive numerical type and *Number* is the wrapper numerical type. An invalid coercion is represented by throw. Such a coercion attempt must result in throwing a ComponentException. Any other coercion error, such as parsing a non-numerical String to a number or inability to coerce a string into a Class or enum object, must be wrapped in a ComponentException and thrown.

source / target	String	Boolean	Character	*Number*	Collection/array
String	v	v.toString()	v.toString()	v.toString()	If v has no elements, null; otherwise the first element of v is coerced.
boolean	Boolean. parseBoolean(v)	v. booleanValue()	v.charValue() != 0	v.*number*Value() != 0	If v has no elements, false; otherwise the first element of v is coerced.
char	v.length() > 0 ? v.charAt(0) : 0	v.booleanValue() ? 1 : 0	v.charValue()	(char) v.*number*Value()	If v has no elements, 0; otherwise the first element of v is coerced.
number	*Number*. parse*Number*(v)	v.booleanValue() ? 1 : 0	(*number*) v.charValue()	v.*number*Value()	
Class	Bundle. loadClass(v)	throw	throw	throw	If v has no elements, null; otherwise the first element of v is coerced.
EnumType	*EnumType*. valueOf(v)	throw	throw	throw	
annotation	throw	throw	throw	throw	throw
array	A single element array is created and v is coerced into it.				An array the size of v is created and each element of v is coerced into it.

Component properties whose names do not map to annotation elements are ignored. If the implementation needs to access these additional properties, it can use a method signature which also receives the properties map in addition to the annotation type.

If there is no corresponding component property for an annotation element, the returned value must be:

- `0` for numerical types and char
- `false` for boolean
- `null` for String, Class, enum and arrays

For elements of an annotation type, a ComponentException must be thrown when calling the element.

This new lifecycle method signatures supporting annotation type arguments are only supported when the component description is declared with namespace `http://www.osgi.org/xmlns/scr/v1.3.0` (or later).

5.6.4 Generation of property information in @Component annotation

The @Component annotation contains two different approaches to define component property values through two different elements: property and properties. In addition, annotations types can now be used to define component property values as well. As annotation elements are unordered when the DS annotations are processed by a tool, the current specification does not define the order of the corresponding elements in the generated component description. However the order in the generated component description defines the order of processing and later property values with the same property name override previously declared ones.

This should be clarified by defining an order of output to the generated component description:

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

2. property element from @Component
3. properties element from @Component

This means that the properties defined through annotation types are declared first in the generated component description, followed by all properties defined through the @Component.property element and finally the properties entries defined through the @Component.properties element.

Since property values defined later in the component description override property values defined earlier in the component description, this means that property values defined in @Component.properties override property values defined in @Component.property which override values defined by default values in annotation types used in lifecycle method signatures.

5.7 Integration with the Configuration Admin Service

DS integrates with the Configuration Admin Service. Therefore SCR must support the latest additions to the Configuration Admin Service.

5.7.1 Targeted PIDs

Section 112.7 Deployment must be updated to refer to 104.3.3 in the Configuration Admin specification and to state that SCR must use the most targeted PID to select the matching Configuration for a component. SCR must also handle the creation and deletion of Configurations with matching targeted PIDs such that:

- The deletion of an in-use Configuration will act as a configuration modification if there is another Configuration with a matching targeted PID.
- The creation of a Configuration with a more precise matching targeted PID will act as a configuration modification.

5.7.2 Multi-location binding

When Java permissions are enabled, SCR must make sure these bindings are properly supported. If the multi-location 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, SCR must check whether the component bundle has the required permission as outlined in Configuration Admin, Section 104.7.6.

5.7.3 Multiple PIDs

DS must be updated to allow a component to specify multiple configuration PIDs for a component. The configuration-pid attribute of the component element will be updated to allow multiple pid values to be specified using a whitespace separated list. For example:

```
<component configuration-pid=”com.acme.host com.acme.system” ...>
```

The configurationPid element of the @Component annotation is updated to support multiple values.

```
String[] configurationPid() default {”$”};
```

This is source compatible for existing code using the @Component annotation even though the type of the element is changed from String to String[]. A special pid string, “$”, can be used with configurationPid() to represent the actual name of the component. The tool processing the annotation into a component description must replace the special string with the actual component name.

When multiple PIDs are specified for a component description, SCR must aggregate the configuration properties from the matching Configurations in the order the PIDs are specified such that properties in Configurations for later specified PIDs take precedence over properties in Configurations for earlier specified PIDs. The creation, updating and deletion for a matching Configuration for any of the PIDs, is a configuration change under section 112.7.1 of the spec and can result in a component configuration becoming satisfied, unsatisfied or modified.

If multiple PIDs are used for a component, only one of the PIDs can be for a factory configuration. If more than one of the PIDs are for factory configurations, this is a configuration error which SCR must log, If this configuration error occurs and the configuration-policy is optional, the component configuration can be satisfied without any of the offending Configurations. If this configuration error occurs and the configuration-policy is required, the component configuration is unsatisfied.

Sections 112.6 and 112.7 must be updated to reflect support for multiple PIDs.

5.8 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).

The introduction of prototype scope services by RFC 195 means we also need to update DS to support this new service feature.

5.8.1 Providing Services

The `servicefactory` attribute on the `service` element is deprecated and replaced by a `scope` attribute supporting the values: `singleton` (default), `bundle` and `prototype`. `servicefactory=false` maps to `scope=singleton` and `servicefactory=true` maps to `scope=bundle`.

This allows SCR to support components being prototype scope services. Since DS never registers the actual component object (that is, even for `scope=singleton`, DS always registers a ServiceFactory to delay component creation and activation), components will never be visible in the service registry with `service.scope=singleton`.

5.8.2 Consuming Services

A `scope` attribute is added to the `reference` element. The `scope` attribute supports the values: `bundle`(default) and `prototype`. When using `scope=bundle`, all references to the service by components in the same bundle will share the same service object. That is, SCR must use BundleContext.getService to obtain the service object. When using `scope=prototype`, each instance of the component will use a difference instance of the service. That is, SCR must use BundleContext.getServiceObjects to obtain the service object and the referenced service must have `service.scope=prototype`. A service without `service.scope=prototype` cannot be used as a bound service for a `scope=prototype` reference since it cannot fulfill the requirement to create multiple service instances for the bundle.

The valid signatures for bind, updated and unbind will be extended to allow ServiceObjects to be injected. For example:

```
        void bind(ServiceObjects);
```

See 5.2 Event Method Signature.

5.8.3 Annotation Changes

The DS Annotations will also be updated to support these new features.

Enum `ServiceScope` is added with values `SINGLETON`, `BUNDLE` and `PROTOTYPE`.

`ServiceScope scope()` is added to the `Component` annotation. `Component.servicefactory()` is deprecated and ignored when `Component.scope()` is specified.

Enum `ReferenceScope` is added with values `BUNDLE` and `PROTOTYPE`.

`ReferenceScope scope()` is added to the `Reference` annotation.

5.8.4 Schema Changes

The DS XML Schema is updated to add a `scope` attribute to the `service` and the `reference` elements. The `servicefactory` attribute of the `service` element is removed since it is replaced by the new `scope` attribute.

5.9 Lookup References

A new LookupReference annotation is defined and a reference element is added to the Component annotation. This allows the use of DS annotation for lookup strategy access to referenced services. The reference element can specify a number of LookupReference annotations from which reference elements can be generated in the component description. The LookupReference annotation is very similar to the existing Reference element but requires the name and service elements to be specified since the LookupReference annotation is not applied to a bind method from whose signature the reference name and service type can be inferred. The LookupReference reference annotation also does not contain updated or unbind elements.

5.10 Minimum Cardinality Property

A new component property is defined called the *minimum cardinality property* of a reference. Like the target property described in 112.6, the minimum cardinality property name is prefixed by the reference name to identify to which reference the minimum cardinality property applies. So the minimum cardinality property name takes the form:

<*refname*>.cardinality.minimum

where <*refname*> is the name of the reference.

The minimum cardinality property can be used by a deployer (e.g. via Configuration Admin) to raise the floor of the cardinality not to exceed the ceiling. That is, a 0..1 cardinality can be raised to a 1..1 cardinality with a minimum cardinality property value of 1. And a 0..n or 1..n cardinality can be raised to a m..n cardinality with a minimum cardinality property value m which must be greater than 0. A 1..1 or 1..n cardinality cannot lowered to a 0..1 or 0..n cardinality since the component developer will have coded to expect at least one bound service. Therefore the minimum cardinality property value must be coercible (see the coercion table above) to a positive integer. If a reference has a minimum cardinality property and its value cannot be coerced into a positive integer, the reference's minimum cardinality property must be ignored. For references with declared cardinalities of 0..1 and 1..1, the only valid value for the reference's minimum cardinality property is 1. Any other value must be ignored.

The minimum cardinality property must be supported for component in older namespaces.

Also, all component property names starting with the reference name following by a period, <*refname*>., are reserved for use by the specification.

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:

```
ComponentDescriptionDTO getComponentDescriptionDTO(Bundle, String)
Collection<ComponentDescriptionDTO> getComponentDescriptionDTOs(Bundle...)
Collection<ComponentConfigurationDTO> getComponentConfigurationDTOs(ComponentDescriptionDTO)
```

See the JavaDoc for details.

7 Java API

OSGi Javadoc

6/3/14 5:33 PM

Package Summary		***Page***
org.osgi.service.component	Service Component Package Version 1.3.	*22*
org.osgi.service.component.annotations	Service Component Annotations Package Version 1.3.	*34*
org.osgi.service.component.runtime	Service Component Runtime Package Version 1.3.	*63*
org.osgi.service.component.runtime.dto	Service Component Runtime Data Transfer Objects Package Version 1.3.	*67*

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.	*23*
ComponentContext	A Component Context object is used by a component instance to interact with its execution context including locating services by reference name.	*26*
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.	*32*
ComponentInstance	A ComponentInstance encapsulates a component instance of an activated component configuration.	*33*

Exception Summary		Page
ComponentException	Unchecked exception which may be thrown by the Service Component Runtime.	*30*

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.	24
String	**COMPONENT_ID** A component property that contains the generated id for a component configuration.	24
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.	23
int	**DEACTIVATION_REASON_BUNDLE_STOPPED** The component configuration was deactivated because the bundle was stopped.	25
int	**DEACTIVATION_REASON_CONFIGURATION_DELETED** The component configuration was deactivated because its configuration was deleted.	25
int	**DEACTIVATION_REASON_CONFIGURATION_MODIFIED** The component configuration was deactivated because its configuration was changed.	25
int	**DEACTIVATION_REASON_DISABLED** The component configuration was deactivated because the component was disabled.	24
int	**DEACTIVATION_REASON_DISPOSED** The component configuration was deactivated because the component was disposed.	25
int	**DEACTIVATION_REASON_REFERENCE** The component configuration was deactivated because a reference became unsatisfied.	24
int	**DEACTIVATION_REASON_UNSPECIFIED** The reason the component configuration was deactivated is unspecified.	24
String	**REFERENCE_TARGET_SUFFIX** The suffix for reference target properties.	24
String	**SERVICE_COMPONENT** Manifest header specifying the XML documents within a bundle that contain the bundle's Service Component descriptions.	23

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.	29
`void`	**`enableComponent`**`(String name)` Enables the specified component name.	28
`org.osgi.framework.BundleContext`	**`getBundleContext`**`()` Returns the `BundleContext` of the bundle which contains this component.	28
`ComponentInstance`	**`getComponentInstance`**`()` Returns the Component Instance object for the component instance associated with this Component Context.	28
`Dictionary<String,Object>`	**`getProperties`**`()` Returns the component properties for this Component Context.	27
`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.	29
`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.	28
`Object`	**`locateService`**`(String name)` Returns the service object for the specified reference name.	27
`Object`	**`locateService`**`(String name, org.osgi.framework.ServiceReference<?> reference)` Returns the service object for the specified reference name and `ServiceReference`.	27
`Object[]`	**`locateServices`**`(String name)` Returns the service objects for the specified reference name.	27

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.

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

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

---

disableComponent

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

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

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

Parameters:
`name` - The name of a component.

---

getServiceReference

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

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

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

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

Class ComponentException

org.osgi.service.component

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

All Implemented Interfaces:
Serializable

```
public class ComponentException
extends RuntimeException
```

Unchecked exception which may be thrown by the Service Component Runtime.

Constructor Summary	Page
`ComponentException(String message)` Construct a new ComponentException with the specified message.	30
`ComponentException(String message, Throwable cause)` Construct a new ComponentException with the specified message and cause.	30
`ComponentException(Throwable cause)` Construct a new ComponentException with the specified cause.	31

Method Summary		Page
`Throwable`	`getCause()` Returns the cause of this exception or `null` if no cause was set.	31
`Throwable`	`initCause(Throwable cause)` Initializes the cause of this exception to the specified value.	31

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.	*32*

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.	*33*
`Object`	`getInstance()` Returns the component instance of the activated component configuration.	*33*

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.	*42*
ReferenceCardinality	Cardinality for the `Reference` annotation.	*53*
ReferencePolicy	Policy for the `Reference` annotation.	*55*
ReferencePolicyOption	Policy option for the `Reference` annotation.	*57*
ReferenceScope	Reference scope for the `Reference` annotation.	*59*
ServiceScope	Service scope for the `Component` annotation.	*61*

Annotation Types Summary		*Page*
Activate	Identify the annotated method as the `activate` method of a Service Component.	*35*
Component	Identify the annotated class as a Service Component.	*36*
Deactivate	Identify the annotated method as the `deactivate` method of a Service Component.	*44*
LookupReference	Define a lookup strategy reference for a `Component`.	*45*
Modified	Identify the annotated method as the `modified` method of a Service Component.	*48*
Reference	Identify the annotated method as a `bind` method of a Service Component.	*49*

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."

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

Required Element Summary		**Page**
String[]	**configurationPid** The configuration PIDs for the configuration of this Component.	40
ConfigurationPolicy	**configurationPolicy** The configuration policy of this Component.	39
boolean	**enabled** Declares whether this Component is enabled when the bundle containing it is started.	38
String	**factory** The factory identifier of this Component.	37
boolean	**immediate** Declares whether this Component must be immediately activated upon becoming satisfied or whether activation should be delayed.	38
String	**name** The name of this Component.	37
String[]	**properties** Property entries for this Component.	39
String[]	**property** Properties for this Component.	39
LookupReference[]	**reference** The lookup strategy references of this Component.	40
ServiceScope	**scope** The service scope for the service of this Component.	40
Class<?>[]	**service** The types under which to register this Component as a service.	37
boolean	**servicefactory** **Deprecated.** *Since 1.3.*	38
String	**xmlns** The XML name space of the Component Description for this Component.	39

Field Detail

NAME

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

Special string representing the name of this Component.

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

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

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

Since:
1.3

Element Detail

name

```
public abstract String name
```

The name of this Component.

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

Default:
""

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

---

service

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

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

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

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

Default:
{}

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

---

factory

```
public abstract String factory
```

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

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

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

---

servicefactory

`public abstract boolean servicefactory`

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

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

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

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

Default:
false
See Also:
"The 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 PIDs for the configuration of this Component.

Each value specifies a configuration PID for this Component.

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

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

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

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

Default:
{ "$" }

Since:
1.2

See Also:
"The configuration-pid attribute of the component element of a Component Description."

---

scope

`public abstract ServiceScope scope`

The service scope for the service of this Component.

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

Default:
ServiceScope.DEFAULT

Since:
1.3

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

---

reference

`public abstract LookupReference[] reference`

The lookup strategy references of this Component.

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

To access references using the event strategy, bind methods are annotated with `Reference`.

Default:
{}
Since:
1.3
See Also:
"The reference 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.	43
OPTIONAL Use the corresponding Configuration object if present but allow the component to be satisfied even if the corresponding Configuration object is not present.	42
REQUIRE There must be a corresponding Configuration object for the component configuration to become satisfied.	43

Method Summary		Page
String	**toString**()	43
static ConfigurationPolicy	**valueOf**(String name)	43
static ConfigurationPolicy[]	**values**()	43

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 LookupReference

org.osgi.service.component.annotations

```
@Retention(value=RetentionPolicy.CLASS)
@Target(value={})
public @interface LookupReference
```

Define a lookup strategy reference for a `Component`.

The referenced service can be accessed using one of the `locateService` methods of `ComponentContext`.

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.

Since:
1.3

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

Required Element Summary		Page
`ReferenceCardinality`	**`cardinality`** The cardinality of the reference.	*46*
`String`	**`name`** The name of this reference.	*45*
`ReferencePolicy`	**`policy`** The policy for the reference.	*46*
`ReferencePolicyOption`	**`policyOption`** The policy option for the reference.	*46*
`ReferenceScope`	**`scope`** The requested service scope for this Reference.	*47*
`Class<?>`	**`service`** The type of the service to bind to this reference.	*45*
`String`	**`target`** The target filter for the reference.	*46*

Element Detail

name

```
public abstract String name
```

The name of this reference.

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.

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."

---

policyOption

`public abstract ReferencePolicyOption policyOption`

The policy option for the reference.

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

Default:
ReferencePolicyOption.RELUCTANT
See Also:
"The policy-option 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
See Also:
    "The scope attribute of the reference 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.	50
String	**name** The name of this reference.	49
ReferencePolicy	**policy** The policy for the reference.	50
ReferencePolicyOption	**policyOption** The policy option for the reference.	51
ReferenceScope	**scope** The requested service scope for this Reference.	51
Class<?>	**service** The type of the service to bind to this reference.	50
String	**target** The target filter for the reference.	50
String	**unbind** The name of the unbind method which is associated with the annotated bind method.	50
String	**updated** The name of the updated method which is associated with the annotated bind method.	51

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.	54
`MANDATORY` The reference is mandatory and unary.	53
`MULTIPLE` The reference is optional and multiple.	54
`OPTIONAL` The reference is optional and unary.	53

Method Summary		Page
`String`	**`toString`**`()`	54
`static ReferenceCardinality`	**`valueOf`**`(String name)`	54
`static ReferenceCardinality[]`	**`values`**`()`	54

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.	55
STATIC The static policy is the most simple policy and is the default policy.	55

Method Summary		Page
String	**toString**()	56
static ReferencePolicy	**valueOf**(String name)	56
static ReferencePolicy[]	**values**()	56

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.	*57*
`RELUCTANT` The reluctant policy option is the default policy option for both `static` and `dynamic` reference policies.	*57*

Method Summary		*Page*
`String`	**`toString`**`()`	*58*
`static ReferencePolicyOption`	**`valueOf`**`(String name)`	*58*
`static ReferencePolicyOption[]`	**`values`**`()`	*58*

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.	59
`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.	59

Method Summary		Page
`String`	**`toString`**`()`	60
`static ReferenceScope`	**`valueOf`**`(String name)`	60
`static ReferenceScope[]`	**`values`**`()`	60

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.	*61*
DEFAULT Default element value for annotation.	*62*
PROTOTYPE When the component is registered as a service, it will be registered as a prototype scope service.	*62*
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.	*61*

Method Summary		*Page*
`String`	**`toString`**`()`	*62*
`static ServiceScope`	**`valueOf`**`(String name)`	*62*
`static ServiceScope[]`	**`values`**`()`	*62*

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 Runtime 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.	*64*

Package org.osgi.service.component.runtime Description

Service Component Runtime 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.runtime; version="[1.3,2.0)"
```

Example import for providers implementing the API in this package:

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

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

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

Since:
1.3

ThreadSafe

Method Summary		Page
void	**`disableComponent`**`(ComponentDescriptionDTO description)` Disables the specified component description.	*66*
void	**`enableComponent`**`(ComponentDescriptionDTO description)` Enables the specified component description.	*65*
`Collection<ComponentConfigurationDTO>`	**`getComponentConfigurationDTOs`**`(ComponentDescriptionDTO description)` Returns the component configurations for the specified component description.	*65*
`ComponentDescriptionDTO`	**`getComponentDescriptionDTO`**`(org.osgi.framework.Bundle bundle, String name)` Returns the `ComponentDescriptionDTO` declared with the specified name by the specified bundle.	*65*
`Collection<ComponentDescriptionDTO>`	**`getComponentDescriptionDTOs`**`(org.osgi.framework.Bundle... bundles)` Returns the component descriptions declared by the specified active bundles.	*64*
boolean	**`isComponentEnabled`**`(ComponentDescriptionDTO description)` Returns whether the specified component description is currently enabled.	*65*

Method Detail

getComponentDescriptionDTOs

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

Returns the component descriptions declared by the specified active bundles.

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

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

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

---

getComponentDescriptionDTO

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

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

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

Parameters:
`bundle` - The bundle declaring the component description. Must not be `null`.
`name` - The name of the component description. Must not be `null`.
Returns:
The declared component description or `null` if the specified bundle is not active or does not declare a component description with the specified name.

---

getComponentConfigurationDTOs

```
Collection<ComponentConfigurationDTO> getComponentConfigurationDTOs(ComponentDescriptionDTO de
scription)
```

Returns the component configurations for the specified component description.

Parameters:
`description` - The component description. Must not be `null`.
Returns:
A collection containing a snapshot of the current component configurations for the specified component description. An empty collection is returned if there are none.

---

isComponentEnabled

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

Returns whether the specified component description is currently enabled.

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

Parameters:
`description` - The component description. Must not be `null`.
Returns:
`true` if the specified component description is currently enabled. Otherwise, `false`.
See Also:
`enableComponent(ComponentDescriptionDTO)`,
`disableComponent(ComponentDescriptionDTO)`,
`ComponentContext.disableComponent(String)`,
`ComponentContext.enableComponent(String)`

---

enableComponent

```
void enableComponent(ComponentDescriptionDTO description)
```

Enables the specified component description.

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

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

Parameters:
`description` - The component description to enable. Must not be `null`.
See Also:
`isComponentEnabled(ComponentDescriptionDTO)`

---

disableComponent

`void` **`disableComponent`**`(ComponentDescriptionDTO description)`

Disables the specified component description.

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

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

Parameters:
`description` - The component description to disable. Must not be `null`.
See Also:
`isComponentEnabled(ComponentDescriptionDTO)`

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

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

Service Component Runtime Data Transfer Objects Package Version 1.3.

See:
Description

Class Summary		Page
BoundReferenceDTO	A representation of a bound reference to a service.	68
ComponentConfigurationDTO	A representation of an actual instance of a declared component description parameterized by component properties.	70
ComponentDescriptionDTO	A representation of a declared component description.	73
ReferenceDTO	A representation of a declared reference to a service.	77

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

Service Component Runtime Data Transfer Objects 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.runtime.dto; version="[1.3,2.0)"
```

Example import for providers implementing the API in this package:

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

Class BoundReferenceDTO

org.osgi.service.component.runtime.dto

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

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

A representation of a bound reference to a service.

Since:
1.3
NotThreadSafe

Field Summary		Page
`String`	`name` The name of the declared reference.	68
`org.osgi.framework.dto.ServiceReferenceDTO[]`	`serviceReferences` The bound services.	69
`String`	`target` The target property of the bound reference.	68

Constructor Summary	Page
`BoundReferenceDTO()`	69

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

Field Detail

name

```
public String name
```

The name of the declared reference.

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

See Also:
`ComponentDescriptionDTO.name`

target

```
public String target
```

The target property of the bound reference.

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

---

serviceReferences

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

The bound services.

Each `org.osgi.framework.dto.ServiceReferenceDTO` in the array represents a service bound to the component configuration.

Constructor Detail

BoundReferenceDTO

```
public BoundReferenceDTO()
```

Class ComponentConfigurationDTO

org.osgi.service.component.runtime.dto

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

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

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

Since:
1.3

NotThreadSafe

Field Summary		Page
`static int`	**`ACTIVE`** The component configuration is active.	*71*
`BoundReferenceDTO[]`	**`boundReferences`** The currently bound references.	*72*
`ComponentDescriptionDTO`	**`description`** The representation of the component configuration's component description.	*71*
`long`	**`id`** The id of the component configuration.	*71*
`Map<String,Object>`	**`properties`** The component properties for the component configuration.	*71*
`static int`	**`SATISFIED`** The component configuration is satisfied.	*71*
`int`	**`state`** The current state of the component configuration.	*71*
`static int`	**`UNSATISFIED`** The component configuration is unsatisfied.	*70*

Constructor Summary	Page
`ComponentConfigurationDTO()`	*72*

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

Field Detail

UNSATISFIED

```
public static final int UNSATISFIED = 1
```

The component configuration is unsatisfied.

This is the initial state of a component configuration. When the component configuration becomes satisfied it enters the `SATISFIED` state.

SATISFIED

```
public static final int SATISFIED = 2
```

The component configuration is satisfied.

Any `services` declared by the component description are registered. If the component configuration becomes unsatisfied for any reason, any declared services must be unregistered and the component configuration returns to the `UNSATISFIED` state.

ACTIVE

```
public static final int ACTIVE = 4
```

The component configuration is active.

This is the normal operational state of a component configuration. The component configuration will move to the `SATISFIED` state when it is deactivated.

description

```
public ComponentDescriptionDTO description
```

The representation of the component configuration's component description.

id

```
public long id
```

The id of the component configuration.

The id is a non-persistent, unique value assigned at runtime. The id is also available as the `component.id` component property.

state

```
public int state
```

The current state of the component configuration.

This is one of `UNSATISFIED`, `SATISFIED` or `ACTIVE`.

properties

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

The component properties for the component configuration.

See Also:

`ComponentContext.getProperties()`

boundReferences

`public BoundReferenceDTO[] boundReferences`

The currently bound references.

Each `BoundReferenceDTO` in the array represents a service bound to a reference of the component configuration. The array will be empty if the component configuration has no bound references.

Constructor Detail

ComponentConfigurationDTO

`public ComponentConfigurationDTO()`

Class ComponentDescriptionDTO

org.osgi.service.component.runtime.dto

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

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

A representation of a declared component description.

Since:
1.3

NotThreadSafe

Field Summary		**Page**
String	**activate** The name of the activate method.	75
org.osgi.framework.dto.BundleDTO	**bundle** The bundle declaring the component description.	74
String[]	**configurationPid** The configuration pids.	76
String	**configurationPolicy** The configuration policy.	76
String	**deactivate** The name of the deactivate method.	75
boolean	**defaultEnabled** The initial enabled state.	74
String	**factory** The component factory name.	74
boolean	**immediate** The immediate state.	75
String	**implementationClass** The fully qualified name of the implementation class.	74
String	**modified** The name of the modified method.	76
String	**name** The name of the component.	74
Map<String,Object>	**properties** The declared component properties.	75
ReferenceDTO[]	**references** The referenced services.	75
String	**scope** The service scope.	74
String[]	**serviceInterfaces** The fully qualified names of the service interfaces.	75

Constructor Summary	**Page**
ComponentDescriptionDTO()	76

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

Field Detail

name

`public String name`

The name of the component.

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

---

bundle

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

The bundle declaring the component description.

---

factory

`public String factory`

The component factory name.

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

---

scope

`public String scope`

The service scope.

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

---

implementationClass

`public String implementationClass`

The fully qualified name of the implementation class.

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

---

defaultEnabled

`public boolean defaultEnabled`

The initial enabled state.

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

---

immediate

`public boolean` **`immediate`**

The immediate state.

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

---

serviceInterfaces

`public String[]` **`serviceInterfaces`**

The fully qualified names of the service interfaces.

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

---

properties

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

The declared component properties.

These are declared in the `property` and `properties` elements.

---

references

`public ReferenceDTO[]` **`references`**

The referenced services.

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

---

activate

`public String` **`activate`**

The name of the activate method.

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

---

deactivate

`public String` **`deactivate`**

The name of the deactivate method.

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

---

modified

`public String` **`modified`**

The name of the modified method.

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

---

configurationPolicy

`public String` **`configurationPolicy`**

The configuration policy.

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

---

configurationPid

`public String[]` **`configurationPid`**

The configuration pids.

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

Constructor Detail

ComponentDescriptionDTO

`public` **`ComponentDescriptionDTO`**`()`

Class ReferenceDTO

org.osgi.service.component.runtime.dto

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

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

A representation of a declared reference to a service.

Since:
1.3

NotThreadSafe

Field Summary		Page
String	**bind** The name of the bind method of the reference.	78
String	**cardinality** The cardinality of the reference.	78
String	**interfaceName** The service interface of the reference.	78
String	**name** The name of the reference.	77
String	**policy** The policy of the reference.	78
String	**policyOption** The policy option of the reference.	78
String	**scope** The scope of the reference.	79
String	**target** The target of the reference.	78
String	**unbind** The name of the unbind method of the reference.	79
String	**updated** The name of the updated method of the reference.	79

Constructor Summary	Page
ReferenceDTO()	79

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

Field Detail

name

```
public String name
```

The name of the reference.

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

---

interfaceName

`public String` **`interfaceName`**

The service interface of the reference.

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

---

cardinality

`public String` **`cardinality`**

The cardinality of the reference.

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

---

policy

`public String` **`policy`**

The policy of the reference.

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

---

policyOption

`public String` **`policyOption`**

The policy option of the reference.

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

---

target

`public String` **`target`**

The target of the reference.

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

---

bind

`public String` **`bind`**

The name of the bind method of the reference.

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

---

unbind

`public String **unbind**`

The name of the unbind method of the reference.

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

---

updated

`public String **updated**`

The name of the updated method of the reference.

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

---

scope

`public String **scope**`

The scope of the reference.

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

Constructor Detail

ReferenceDTO

`public **ReferenceDTO**()`

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.

8.6 New Life Cycle States

The following proposed new states were removed per bug 2567.

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.

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, although 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

Name	BJ Hargrave
Company	IBM
e-mail	hargrave@us.ibm.com

10.3 Acronyms and Abbreviations

10.4 End of Document