RFC 134 Declarative Services Updates

Final

12 Pages

Abstract

This RFC specifies some minor changes requested for Declarative Services.

0 Document Information

0.1 Table of Contents

0.2 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 6.1.

`Source code is shown in this typeface.`

0.3 Revision History

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

Revision	Date	Comments
Initial	2008 Mar 11	Initial Draft BJ Hargrave, hargrave@us.ibm.com
2nd draft	2008 Mar 16	Updated based upon feedback from CPEG BJ Hargrave, hargrave@us.ibm.com
3rd draft	2008 Mar 27	Updated based upon feedback from CPEG BJ Hargrave, hargrave@us.ibm.com
4th draft	2008 Apr 24	Updated based upon bug 600. BJ Hargrave, hargrave@us.ibm.com
5th draft	2008 Oct 20	Updated based upon bug 780 BJ Hargrave

1 Introduction

Some minor updated to the Declarative Services Specification have been requested since it was released. This RFC specifies those changes.

2 Application Domain

This RFC defines changes to section 112, Declarative Services, of the specification.

3 Technical Solution

3.1 Bug 144: No component instance if no Configuration

A way is needed for the component declaration to say only create a component configurationIF there is a Configuration(or Configurations).

To support this, the follow attribute is added to the component element:

```
<attribute name="configuration-policy" type="scr:Tconfiguration-policy"
           default="optional" use="optional" />
<simpleType name="Tconfiguration-policy">
           <restriction base="string">
                      <enumeration value="optional" />
                      <enumeration value="require" />
                      <enumeration value="ignore" />
           </restriction>
</simpleType>
```

If the attribute is present and set to require, then a component cannot be satisfied (section 112.5.2) unless there is a Configuration in ConfigurationAdmin for the component. In this situation, the *No Configuration* case in 112.7 does not apply.If the component is a Factory Component and the component is not satisfied because there is no Configuration present, then the ComponentFactory service will not be registered.

If the attribute is present and set to ignore, then ConfigurationAdmin will not be consulted for the component. In this situation, only the *No Configuration* case in 112.7 applies.

If the attribute is not present or present and set to optional, then SCR will act as it did prior to this RFC. That is, a Configuration will be used if present in ConfigurationAdmin.

3.2 Bugs 244 and 780: Extend SCR to allow alternate method signatures

A way is needed to avoid using DS API at all in components with SCR. This means the activate and deactivate methods should not require the ComponentContext parameter. We should also allow the names of the activate and deactivate methods to be specified to avoid requiring specific method names.

To support this, the follow attributes will be added to the component element:

```
<attribute name="activate" type="token" use="optional" default="activate" />
<attribute name="deactivate" type="token" use="optional default="deactivate" />
```

The activate attribute will specify the name of the activate method and the deactivate attribute will specify the name of the deactivate method.

The signature for the activate and deactivate methods is:

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

<arguments> can be zero or more arguments.

For the activate method each argument must be of one of the following types:

- ComponentContext – the Component Context for the component
- BundleContext – the Bundle Context of the component's bundle
- Map – the Component Properties from ComponentContext.getProperties.

If any argument of the activate method is not one of the above types, SCR must log an error message with the Log Service, if present, and the component configuration is not activated.

For the deactivate method each argument must be of one of the following types:

- int/Integer – the deactivation reason
- ComponentContext – the Component Context for the component
- BundleContext – the Bundle Context of the component's bundle
- Map – the Component Properties from ComponentContext.getProperties.

If any argument of the deactivate method is not one of the above types, SCR must log an error message with the Log Service, if present, and the deactivation of the component configuration will continue.

The methods may also be declared public. The same rules as specified in 112.5.8 will be used to locate the activate and deactivate methods in the implementation class hierarchy.

3.2.1 Component deactivation reasons

When a component is deactivated, the reason for the deactivation can be passed to the deactivate method. The following deactivation reasons are specified in `ComponentConstants`.

```
            /**
             * The reason the component instance was deactivated is unspecified.
             *
             * @since 1.1
             */
            public static final int DEACTIVATION_REASON_UNSPECIFIED = 0;

            /**
             * The component instance was deactivated because the component was
disabled.
             *
             * @since 1.1
             */
            public static final int DEACTIVATION_REASON_DISABLED = 1;

            /**
             * The component instance was deactivated because a reference became
unsatisfied.
             *
             * @since 1.1
```

```
     */
    public static final int DEACTIVATION_REASON_REFERENCE = 2;

    /**
     * The component instance was deactivated because its configuration was
changed.
     *
     * @since 1.1
     */
    public static final int DEACTIVATION_REASON_CONFIGURATION_MODIFIED = 3;

    /**
     * The component instance was deactivated because its configuration was
deleted.
     *
     * @since 1.1
     */
    public static final int DEACTIVATION_REASON_CONFIGURATION_DELETED = 4;

    /**
     * The component instance was deactivated because the component was
disposed.
     *
     * @since 1.1
     */
    public static final int DEACTIVATION_REASON_DISPOSED = 5;

    /**
     * The component instance was deactivated because the bundle was stopped.
     *
     * @since 1.1
     */
    public static final int DEACTIVATION_REASON_BUNDLE_STOPPED = 6;
```

3.2.2 Allow bind and unbind methods to receive the service properties

A bind or unbind method must have one of the following prototypes:

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

If the bind or unbind method has the third prototype, then the service object of the bound service is passed to the method as the first argument and a Map containing the service properties of the bound service is passed as the second argument. The method's first parameter type must be assignable from the type specified by the reference's interface attribute. That is, the service object of the bound service must be castable to the method's first parameter type.

When searching for the bind or unbind method to call, SCR must look through the component implementation class hierarchy. The declared methods of each class are searched for a method with the specified name that takes one or two parameters. The method is searched for using the following priority:

1.The method takes a single parameter and the type of the parameter is org.osgi.framework.ServiceReference.

2.The method takes a single parameter and the type of the parameter is the type specified by the reference's interface attribute.

3.The method takes a single parameter and the type of the parameter 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.

4.The method takes two parameters and the type of the first parameter is the type specified by the reference's interface attribute and the type of the second parameter is java.util.Map

5.The method takes two parameters and the type of the first parameter is assignable from the type specified by the reference's interface attribute and the type of the second parameter is 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.

3.1 Bug 567: Allow use of wildcards in Service-Component header

A way is needed to allow wild specification of the XML documents containing the component descriptions.

To support this, section 112.4.1 will be updated to state that the path element of the Service-Component header grammar may include wildcards in the last component of the path. For example:

```
Service-Component: OSGI-INF/*.xml
```

Only the last component of the path may use wildcards so that Bundle.findEntries can be used to locate the XML document within the bundle and its fragments.

3.2 Bug 600: Making name attributes optional

To reduce the amount of XML that must be written for a component description, the `name` attributes of the `component` and `reference` elements will be changed from required to optional. This change is only effective for documents in the new namespace.

The default value of the `name` attribute of the `component` element is the value of the `class` attribute of the nested `implementation` element.

The default value of the `name` attribute of the `reference` element is the value of the `interface` attribute of the `reference` element.

3.3 XML schema change

The schema has been updated and thus has a new namespace

http://www.osgi.org/xmlns/scr/v1.1.0

This is due to changes to support backwards and forwards computability in the future (after this namespace change). Hopefully we can avoid further namespace changes by introducing extra requirements on the SCR parser at this time.

3.3.1 Updated schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
       xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
```

```
targetNamespace="http://www.osgi.org/xmlns/scr/v1.1.0"
version="1.1.0">

<annotation>
	<documentation xml:lang="en">
		This is the XML Schema for component descriptions used by
		the Service Component Runtime (SCR). Component description
		documents may be embedded in other XML documents. SCR will
		process all XML documents listed in the Service-Component
		manifest header of a bundle. XML documents containing
		component descriptions may contain a single, root component
		element or one or more component elements embedded in a
		larger document. Use of the namespace is optional if the
		document only contains a root component element. In this
		case, the scr namespace is assumed. Otherwise the namespace
		must be specified.
	</documentation>
</annotation>
<element name="component" type="scr:Tcomponent" />
<complexType name="Tcomponent">
	<annotation>
		<documentation xml:lang="en">
			Implementations of SCR must not require component
			descriptions to specify the subelements of the component
			element in the order as required by the schema. SCR
			implementations must allow other orderings since
			arbitrary orderings do not affect the meaning of the
			component description. Only the relative ordering of
			property and properties element have meaning.
		</documentation>
	</annotation>
	<sequence>
		<choice minOccurs="0" maxOccurs="unbounded">
			<element name="property" type="scr:Tproperty" />
			<element name="properties" type="scr:Tproperties" />
		</choice>
		<element name="service" type="scr:Tservice" minOccurs="0"
			maxOccurs="1" />
		<element name="reference" type="scr:Treference"
			minOccurs="0" maxOccurs="unbounded" />
		<element name="implementation" type="scr:Timplementation"
			minOccurs="1" maxOccurs="1" />
		<any namespace="##any" processContents="lax" minOccurs="0"
			maxOccurs="unbounded" />
	</sequence>
	<attribute name="enabled" type="boolean" default="true"
		use="optional" />
	<attribute name="name" type="token" use="optional">
		<annotation>
			<documentation xml:lang="en">
				The default value of this attribute is the value of
				the class attribute of the nested implementation
```

```
                        element.
                    </documentation>
                </annotation>
            </attribute>
            <attribute name="factory" type="string" use="optional" />
            <attribute name="immediate" type="boolean" use="optional" />
            <attribute name="configuration-policy"
                type="scr:Tconfiguration-policy" default="optional" use="optional" />
            <attribute name="activate" type="token" use="optional"
                default="activate" />
            <attribute name="deactivate" type="token" use="optional"
                default="deactivate" />
            <anyAttribute />
        </complexType>
        <complexType name="Timplementation">
            <sequence>
                <any namespace="##any" processContents="lax" minOccurs="0"
                    maxOccurs="unbounded" />
            </sequence>
            <attribute name="class" type="token" use="required" />
            <anyAttribute />
        </complexType>
        <complexType name="Tproperty">
            <simpleContent>
                <extension base="string">
                    <attribute name="name" type="string" use="required" />
                    <attribute name="value" type="string" use="optional" />
                    <attribute name="type" type="scr:Tjava-types"
                        default="String" use="optional" />
                    <anyAttribute />
                </extension>
            </simpleContent>
        </complexType>
        <complexType name="Tproperties">
            <sequence>
                <any namespace="##any" processContents="lax" minOccurs="0"
                    maxOccurs="unbounded" />
            </sequence>
            <attribute name="entry" type="string" use="required" />
            <anyAttribute />
        </complexType>
        <complexType name="Tservice">
            <sequence>
                <element name="provide" type="scr:Tprovide" minOccurs="1"
                    maxOccurs="unbounded" />
                <!-- It is non-deterministic, per W3C XML Schema 1.0:
http://www.w3.org/TR/xmlschema-1/#cos-nonambig
                    to use namespace="##any" below. -->
                <any namespace="##other" processContents="lax" minOccurs="0"
                    maxOccurs="unbounded" />
            </sequence>
            <attribute name="servicefactory" type="boolean" default="false"
```

```
                use="optional" />
        <anyAttribute />
</complexType>
<complexType name="Tprovide">
        <sequence>
                <any namespace="##any" processContents="lax" minOccurs="0"
                        maxOccurs="unbounded" />
        </sequence>
        <attribute name="interface" type="token" use="required" />
        <anyAttribute />
</complexType>
<complexType name="Treference">
        <sequence>
                <any namespace="##any" processContents="lax" minOccurs="0"
                        maxOccurs="unbounded" />
        </sequence>
        <attribute name="name" type="token" use="optional">
                <annotation>
                        <documentation xml:lang="en">
                                The default value of this attribute is the value of
                                the interface attribute of this element. If multiple
                                instances of this element within a component element
                                use the same value for the interface attribute, then
                                using the default value for this attribute will result
                                in duplicate names. In this case, this attribute
                                should be specified with a unique value.
                        </documentation>
                </annotation>
        </attribute>
        <attribute name="interface" type="token" use="required" />
        <attribute name="cardinality" type="scr:Tcardinality"
                default="1..1" use="optional" />
        <attribute name="policy" type="scr:Tpolicy" default="static"
                use="optional" />
        <attribute name="target" type="string" use="optional" />
        <attribute name="bind" type="token" use="optional" />
        <attribute name="unbind" type="token" use="optional" />
        <anyAttribute />
</complexType>
<simpleType name="Tjava-types">
        <restriction base="string">
                <enumeration value="String" />
                <enumeration value="Long" />
                <enumeration value="Double" />
                <enumeration value="Float" />
                <enumeration value="Integer" />
                <enumeration value="Byte" />
                <enumeration value="Character" />
                <enumeration value="Boolean" />
                <enumeration value="Short" />
        </restriction>
</simpleType>
```

```
	<simpleType name="Tcardinality">
		<restriction base="string">
			<enumeration value="0..1" />
			<enumeration value="0..n" />
			<enumeration value="1..1" />
			<enumeration value="1..n" />
		</restriction>
	</simpleType>
	<simpleType name="Tpolicy">
		<restriction base="string">
			<enumeration value="static" />
			<enumeration value="dynamic" />
		</restriction>
	</simpleType>
	<simpleType name="Tconfiguration-policy">
		<restriction base="string">
			<enumeration value="optional" />
			<enumeration value="require" />
			<enumeration value="ignore" />
		</restriction>
	</simpleType>
	<attribute name="must-understand" type="boolean">
		<annotation>
			<documentation xml:lang="en">
				This attribute should be used by extensions to documents
				to require that the document consumer understand the
				extension.
			</documentation>
		</annotation>
	</attribute>
</schema>
```

4 Considered Alternatives

None at this time.

5 Security Considerations

These changes do not affect the security of the specification.

6 Document Support

6.1 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

6.1 Author's Address

Name	BJ Hargrave
Company	IBM Corporation
Address	800 N Magnolia Av, Orlando, FL 32803
Voice	+1 386 848 1781
e-mail	hargrave@us.ibm.com

6.2 Acronyms and Abbreviations

6.3 End of Document