



Service Component Architecture Spring Component Implementation Specification Version 1.1

Committee Specification Draft 01

23 May 2011

Specification URIs:

This version:

<http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/csd01/sca-springci-spec-v1.1-csd01.doc>
<http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/csd01/sca-springci-spec-v1.1-csd01.pdf> (Authoritative)
<http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/csd01/sca-springci-spec-v1.1-csd01.html>

Previous version:

N/A

Latest version:

<http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/sca-springci-spec-v1.1.doc>
<http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/sca-springci-spec-v1.1.pdf>
(Authoritative)
<http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/sca-springci-spec-v1.1.html>

Technical Committee:

OASIS Service Component Architecture / J (SCA-J) TC

Chairs:

David Booz, IBM
Anish Karmarkar, Oracle

Editors:

David Booz, IBM
Mike Edwards, IBM
Anish Karmarkar, Oracle

Related work:

This specification replaces or supersedes:

- [Service Component Architecture Spring Component Implementation Specification Version 1.00](#), March 21 2007

This specification is related to:

- [Service Component Architecture Assembly Model Specification Version 1.1](#)
- [SCA Policy Framework Version 1.1](#)
- XML schemas: [sca-springci-spec/v1.1/csd01/xsd/](#)

Declared XML namespaces:

<http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810>

<http://docs.oasis-open.org/ns/opencsa/sca/200912>

Abstract:

The SCA Spring component implementation specification specifies how the Spring Framework **[SPRING]** can be used with SCA. This specification extends the SCA Assembly Model by defining how a Spring Framework application context provides an implementation of an SCA component, including its various attributes such as services, references, and properties and how that application context is used in SCA as a component implementation type. The goals of this specification are:

Coarse-grained integration: The integration with Spring is at the SCA Component level, where a Spring application context provides a component implementation, exposing services and using references via SCA. This means that a Spring application context defines the internal structure of an implementation.

Start from SCA Component Type: Use of Spring Framework to implement any SCA Component that uses WSDL or Java interfaces to define services, possibly with some SCA specific extensions.

Start from Spring context: Generation of an SCA Component from any Spring application context and use that component within an SCA assembly.

Status:

This document was last revised or approved by the OASIS Service Component Architecture / J (SCA-J) TC on the above date. The level of approval is also listed above. Check the "Latest Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/sca-j/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/sca-j/ipr.php>).

Citation format:

When referencing this specification the following citation format should be used:

[SCA-Spring-CI-v1.1]

Service Component Architecture Spring Component Implementation Specification Version 1.1. 23 May 2011. OASIS Committee Specification Draft 01. <http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/csd01/sca-springci-spec-v1.1-csd01.pdf>.

Notices

Copyright © OASIS Open 2011. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](#) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

Table of Contents

1	Introduction.....	5
1.1	Terminology	5
1.2	Normative References	5
1.3	Non-Normative References	5
1.4	Testcases.....	6
2	Spring application context as component implementation	7
2.1	Structure of a Spring Application Context.....	8
2.1.1	Spring Beans	9
2.1.2	Property and Constructor Argument References	10
2.2	Direct use of SCA references within a Spring configuration.....	10
2.3	Explicit declaration of SCA related beans inside a Spring Application Context	11
2.3.1	SCA Service element	11
2.3.2	SCA Reference element.....	12
2.3.3	SCA Property element.....	13
2.3.4	Example of a Spring Application Context with SCA Spring Extension Elements	14
2.3.5	Example of a Spring Application Context without Extension Elements	14
2.4	Handling multiple application contexts in SCA runtime	15
3	Component Type of a Spring Application Context	17
3.1	Introspecting the Type Implied by a Spring Bean Reference.....	20
4	Specifying the Spring Implementation Type in an Assembly	21
5	Conformance	23
5.1	SCA Spring Component Implementation Composite Document.....	23
5.2	SCA Spring Application Context Document	23
5.3	SCA Runtime	23
A.	XML Schemas	24
A.1	sca-implementation-spring.xsd	24
A.2	SCA Spring Extension schema - sca-spring-extensions.xsd	24
B.	Conformance Items	26
C.	Acknowledgements	28
D.	Revision History.....	30

1 Introduction

The SCA Spring component implementation specification specifies how the Spring Framework **[SPRING]** can be used with SCA. This specification extends the SCA Assembly Model by defining how a Spring Framework application context provides an implementation of an SCA component, including its various attributes such as services, references, and properties and how that application context is used in SCA as a component implementation type. The goals of this specification are:

Coarse-grained integration: The integration with Spring is at the SCA Component level, where a Spring application context provides a component implementation, exposing services and using references via SCA. This means that a Spring application context defines the internal structure of a component implementation.

Start from SCA Component Type: Use of Spring Framework to implement any SCA Component that uses WSDL or Java interfaces to define services, possibly with some SCA specific extensions.

Start from Spring context: Generation of an SCA Component from any Spring context and use that component within an SCA assembly.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **[RFC2119]**.

1.2 Normative References

- [RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997.
<http://www.ietf.org/rfc/rfc2119.txt>
- [SCA-ASSEMBLY]** OASIS Committee Draft 06, "SCA Assembly Model Specification V1.1", August 2010
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd06.pdf>
- [SCA-POLICY]** OASIS Committee Draft 04, "SCA Policy Framework Specification Version 1.1", September 2010
<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd04.pdf>
- [JAVA-CAA]** OASIS Committee Draft 05, "SCA Java Common Annotations and APIs Specification Version 1.1", November 2010
<http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-csd05.pdf>
- [SPRING]** Spring Framework Specification
<http://static.springsource.org/spring/docs/2.5.x/reference/index.html>

1.3 Non-Normative References

- [SCA-SPRING-TC]** OASIS Working Draft 01, "TestCases for the Spring Component Implementation Specification V1.1", April 2011
<http://www.oasis-open.org/committees/download.php/41746/sca-springci-1.1-testcases-wd01.odt>

44 **1.4 Testcases**

45 The TestCases for the SCA Spring Component Implementation Specification Version 1.1 [**SCA-SPRING-**
46 **TC**] defines the TestCases for the SCA-J Spring Component Implementation specification. The TestCases
47 represent a series of tests that SCA runtimes are expected to pass in order to claim conformance to the
48 requirements of the SCA-J Spring Component Implementation specification.

2 Spring application context as component implementation

A Spring Application Context can be used as an implementation of an SCA component.

Conceptually, this can be represented as follows:

Figure 1 below illustrates an SCA composite containing two components, both of which are implemented by Spring application contexts.

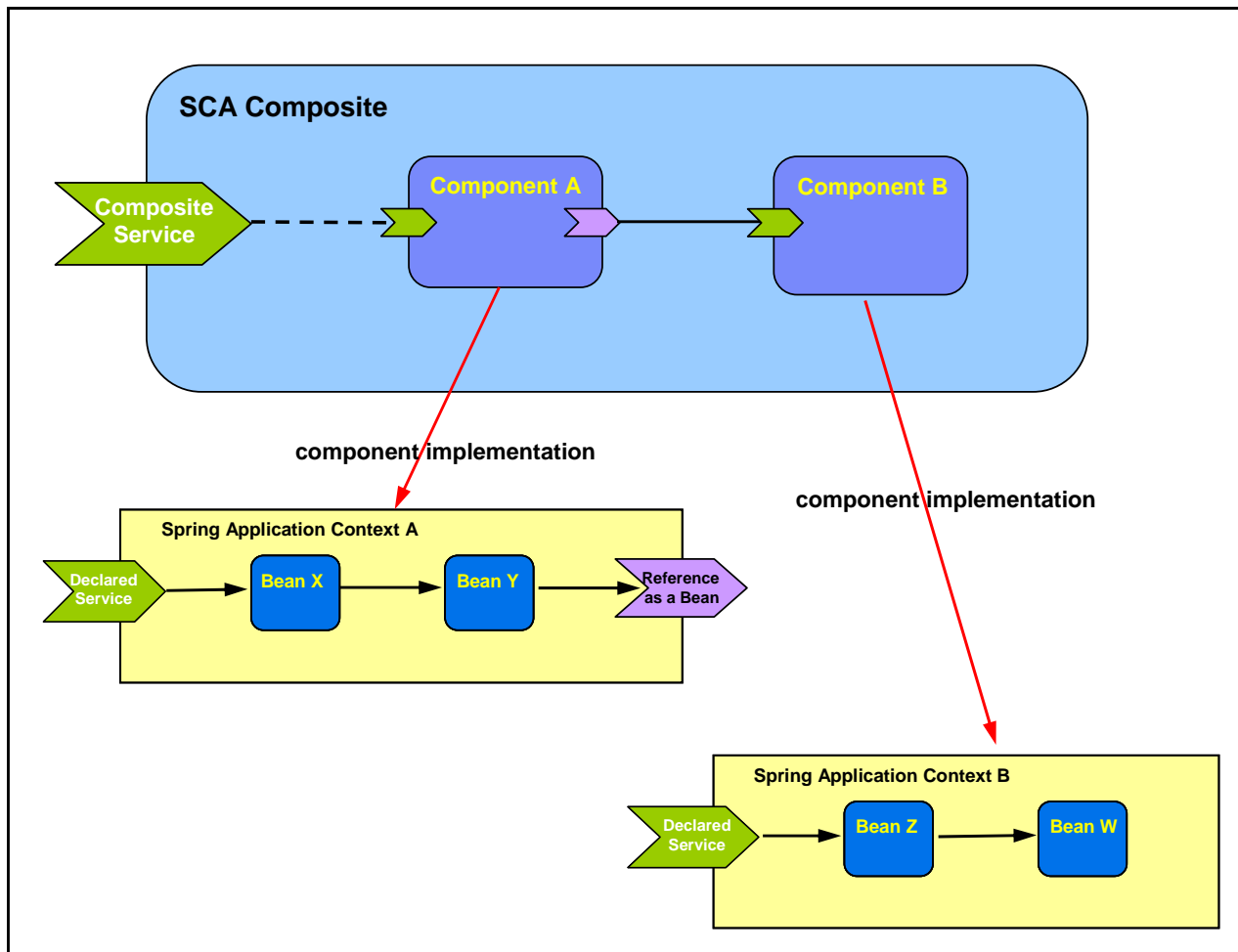
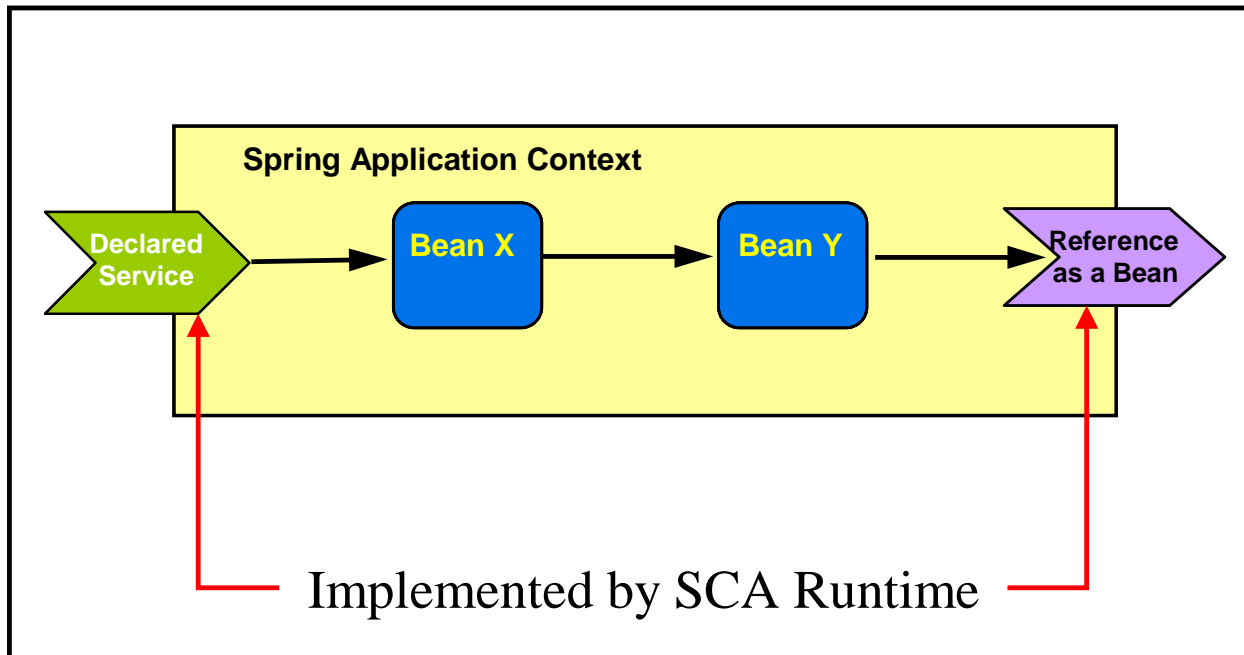


Figure 1 SCA Domain with two Spring application contexts as component implementations

Each component has one declared service. Component A is implemented by an application context, Context A, composed of two Spring beans. Here, bean X is exposed as an SCA service. Bean Y has a reference to an external SCA service. This service reference is wired to the service offered by the second component, Component B which is also implemented by another Spring context, Context B, which has a single declared service, which is provided by Bean Z.

A component that uses Spring for an implementation can wire SCA services and references without introducing SCA metadata into the Spring configuration. The Spring context does not need to know anything about the SCA environment. All binding and policy declarations occur in the SCA runtime implementation and does not enter into the Spring space.



69

70 *Figure 2 Relationship of Spring Application Context artifacts to SCA Services & References*

71 Figure 2 shows two of the points where the SCA runtime interacts with the Spring context: services and
 72 references. Each service is offered by one of the Spring beans within the application context, while each
 73 reference is a property dependency of one or more of the Spring beans in the application context which is
 74 not satisfied by another bean. Services and references can be introspected from a Spring application
 75 context which has no SCA specific features. However, for greater control, it is possible to annotate the
 76 application context file with SCA-specific extensions, for example this can be done to limit the number of
 77 services offered by a particular application context.

78 Any policy specification is done in the SCA composite - and this extends to the running application where
 79 service calls into the Spring application context have policy related processing performed by the SCA
 80 runtime (e.g. decryption of encrypted messages) before the final message is delivered to the target
 81 Spring bean. In the same way, on outbound service invocations from the application context, references
 82 supplied by the SCA runtime can provide policy implementation.

83 2.1 Structure of a Spring Application Context

84 Spring [**SPRING**] applications are described by a declarative XML file called a Spring Application
 85 Context. The structure of the parts of a Spring Application context relevant to SCA is outlined in the
 86 following pseudo-schema

```
87 <beans>
88   <bean id="xs:string" name="xs:string" class="xs:string"
89     scope="xs:string">*
90     <property name="xs:string" value="xs:string"? ref="xs:string"?>*
91       <value type="xs:string"?/>?
92       <bean/>?
93       <ref bean="xs:string"? local="xs:IDREF"? parent="xs:string"?/>?
94       <idref bean="xs:string" local="xs:IDREF"?/>?
95       <list/>?
96       <map/>?
97       <set/>?
98       <lookup-method/>?
99       <replaced-method/>?
100  </property>
```



```

101     <constructor-arg ref="xs:string"? index="xs:string"
102         type="xs:string"? value="xs:string"?>*
103     <value/>?
104     <bean/>?
105     <ref bean="xs:string"/>?
106     <idref bean="xs:string"/>?
107     <list/>?
108     <map/>?
109     <set/>?
110     <props/>?
111 </constructor-arg>
112 <meta/>*
113 <qualifier/>*
114 <lookup-method/>*
115 <replaced-method/>*
116 <any/>*
117 </bean>
118 </beans>

```

119 Example 1: Pseudo-schema for the Spring Application Context

120 2.1.1 Spring Beans

121 The application context consists of a set of <bean/> definitions, where each bean is a Java class that can
122 offer service(s) which are available for use by other beans - and in the context of SCA, a bean can
123 become an SCA service of the component that uses the Spring application context as its implementation.

124 The Java class of a <bean/> is defined by its @class attribute.

125 2.1.1.1 Bean ID & Name

126 A <bean/> can be given either zero or one ID, and can be given zero or more names, using its @id and
127 @name attributes. These names have to be unique within the application context. The id and names can
128 be used to refer to the bean, for example, when one bean has a dependency on another bean.

129 However, it is possible for a bean to have no ID and no names. From an SCA perspective, such
130 anonymous beans are purely for use within the application context - anonymous beans cannot be used
131 for an SCA service, for example.

132 2.1.1.2 Inner Beans

133 As can be seen from the pseudo-schema in Example 1, it is possible to nest a <bean/> within another
134 <bean/> declaration. Nested beans of this kind are termed **inner beans**. Inner beans are purely for use
135 within the application context and have no direct relationship with SCA.

136 2.1.1.3 Bean Properties

137 A <bean/> can have zero or more <property/> subelements. Each <property/> represents a dependency
138 of the bean class, which have to be injected into the class when it is instantiated. Injection is typically be
139 means of a setter method on the bean class.

140 From a Spring perspective, the property value is simply a Java primitive or Java class that is needed by
141 the bean class. From an SCA perspective, a property could be an SCA property or a property could be an
142 SCA reference to a target service, depending on the type of the <property/>.

143 2.1.1.4 Bean Constructor Arguments

144 A <bean/> can have zero or more <constructor-arg/> subelements. These elements are very similar to
145 <property/> elements in that they represent a dependency of the bean class, which have to be injected
146 into the class when it is instantiated. The difference between <constructor-arg/> elements and

147 <property/> elements is that <constructor-arg/> values are injected into the class through parameters on
148 the bean class constructor method, rather than through setter methods.

149 **2.1.2 Property and Constructor Argument References**

150 <property/> and <constructor-arg/> elements can supply their dependencies "by value", through data held
151 directly within the element, by means of the @value attribute, the <value/> subelement or the <bean/>
152 subelement.

153 Collections can be supplied to a bean class by means of the <list/>, <set/> and <map/> subelements.

154 Of relevance to SCA are <property/> and <constructor-arg/> elements that supply their dependencies "by
155 reference", where they contain references to data supplied elsewhere. Typically, these references are to
156 other <bean/> elements in the same application context. However, when using a Spring application
157 context within an SCA environment, the references can be to SCA references and SCA properties,
158 configured by the SCA component using the application context as its implementation.

159 References are made using the @ref attribute and the <ref/> and <idref/> subelements of <property/>
160 and <constructor-arg/> elements. It is also possible to have references within collections, since <list/>,
161 <set/> and <map/> subelements can contain <ref/> and <idref/> entries.

162 Each @ref attribute, <ref/> element or <idref/> identifies another bean within the application context, via
163 its ID or its one of its names.

164 For SCA, it is possible to have references of this type mapped to SCA references or SCA properties,
165 simply by means of having those references left "dangling" - ie not pointing to any bean within the
166 application context. Alternatively, SCA references and SCA properties can be explicitly modelled within
167 the Spring application context using extension elements, as described in the section "[Explicit declaration
168 of SCA related beans inside a Spring Application Context](#)".

169 **2.2 Direct use of SCA references within a Spring configuration**

170 The SCA runtime hosting the Spring application context implementing a component creates a parent
171 application context in which all SCA references are defined as beans using the SCA reference name as
172 the bean name. These beans are automatically visible in the child (user application) context.

173 The following Spring configuration provides a model for Spring application context A, expressed in figure
174 1 above. In this example, there are two Spring beans, X and Y. The Spring bean named "X" provides a
175 service which is configured and invoked via SCA and Spring bean Y contains a reference to a service
176 which is supplied by SCA.

```
177     <beans>  
178         <bean id="X" class="org.xyz.someapp.SomeClass">  
179             <property name="foo" ref="Y"/>  
180         </bean>  
181         <bean id="Y" class="org.xyz.someapp.SomeOtherClass">  
182             <property name="bar" ref="SCAReference"/>  
183         </bean>  
184     </beans>
```

185 Two beans are defined. The bean named "X" contains one property (i.e. reference) named "foo" which
186 refers to the second bean in the context, named "Y". The bean "Y" also has a single property named
187 "bar", which refers to the SCA service reference, with the name "SCAReference"

188 The SCA composite contains service and reference definitions for a component that uses the Spring
189 application context as its implementation, with appropriate binding information:

```
190     <composite name="BazComposite">  
191         <component name="SpringComponent">  
192             <implementation.spring location="..."/>
```

```
193     <service name="X"/>
194     <reference name="SCARreference" .../> <!-- binding info specified -->
195 </component>
196 </composite>
```

197 The only part of this that is specific to Spring is the `<implementation.spring>` element. The
198 `location` attribute of that element specifies the Spring application context file(s) to use, either as a
199 direct pointer to a single file, or via a reference to an archive file or a directory that contains one or more
200 Spring application context files (see the section "Specifying the Spring Implementation Type in an
201 Assembly" for more details).

202 Each `<service>` element used with `<implementation.spring>` by default includes the name of the
203 Spring bean that is to be exposed as an SCA service in its name attribute. So, for Spring, the name
204 attribute of a service plays two roles: it identifies a Spring bean, and it names the service for the
205 component. The service element above has a name of "X", so there is a Spring bean with that name.
206 The SCA component also contains a `<reference>` element named "SCARreference". The reference name
207 becomes an addressable name within the Spring application context – so, in this case, "SCARreference"
208 can be referred to by bean "Y" in the Spring configuration above.

209 The SCA runtime is responsible for setting up the references and exposing them as beans with their
210 indicated names in the spring context. This is usually accomplished by creating a parent context which
211 has the appropriate beans defined and the context supplied by the implementation becomes the child of
212 this context. Thus, the references – e.g. the "SCARreference" that bean "Y" uses for its "bar" property –
213 are available to the context.

214 2.3 Explicit declaration of SCA related beans inside a Spring 215 Application Context

216 It is possible to explicitly declare SCA-related beans inside a Spring application context. A bean within the
217 application context can be declared to be an SCA service. References to beans made within the
218 application context can be declared to be either SCA properties or SCA references.

219 These capabilities are provided by means of a set of SCA extension elements, which can be placed
220 within a Spring application context. The SCA extension elements are declared in the SCA Spring
221 Extension schema - `sca-spring-extensions.xsd` - which is shown in Appendix A. SCA extension elements
222 within a Spring application context MUST conform to the SCA Spring Extension schema declared in `sca-`
223 `spring-extensions.xsd`. [SPR20006]

224 For example, to declare a bean that represents the service referred to by an SCA reference named
225 "SCARreference" the following is declared in the application context:

```
226     <sca:reference name="SCARreference" type="com.xyz.SomeType"/>
```

227 The SCA Spring extension elements are:

- 228 • **<sca:reference>** This element defines a Spring bean representing an SCA service which is
229 external to the Spring application context.
- 230 • **<sca:property>** This element defines a Spring bean which represents a property of the SCA
231 component which configures the Spring composite.
- 232 • **<sca:service>** This element defines a bean that the Spring composite exposes as an SCA
233 service.

234 2.3.1 SCA Service element

235 The SCA service element declares a service that is offered by the Spring application context as an SCA
236 service. When an application context contains one or more SCA service elements, these elements
237 declare all the services that are made available by the application context when it is used as a component
238 implementation. In this way, the service elements provide the developer with a means to control which
239 Spring beans are exposed as SCA services - if no SCA service elements are present in the application
240 context, the default behaviour is to expose all the Spring beans as SCA services.

241 The SCA service element can also declare other attributes of the SCA service. In particular, policy can
242 be associated with the service using the @requires and @policySets attributes.

243 The pseudo-schema for the service element is:

```
244 <beans xmlns="http://www.springframework.org/schema/beans"
245       xmlns:xs="http://www.w3.org/2001/XMLSchema"
246       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
247       xmlns:sca=
248         "http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810"
249
250     ...
251     <sca:service name="xs:NCName"
252               type="xs:NCName"?
253               target="xs:NCName"
254               requires="list of xs:QName"?
255               policySets="list of xs:QName"?/>
256     ...
257
258 </beans>
```

259 The **service** element has the following **attributes**:

- 260 • **name : NCName (1..1)** - the name of the service. The value of the @name attribute of an
261 <sca:service/> subelement of a <beans/> element MUST be unique amongst the <service/>
262 subelements of the <beans/> element. [SPR20001]
- 263 • **type : NCName (0..1)** - the type of the service, declared as the fully qualified name of a Java
264 class. If omitted, the type of the service is introspected from the Spring bean class identified by
265 the @target attribute.
- 266 • **target : NCName (1..1)** - the name of a <bean/> element within the application context which
267 provides the service declared by the sca:service element. The @target attribute of a <service/>
268 subelement of a <beans/> element MUST have the value of the @name attribute of one of the
269 <bean/> subelements of the <beans/> element. [SPR20002]
- 270 • **requires : QName (0..1)** - a list of policy intents. See the [Policy Framework specification](#)
271 [\[POLICY\]](#) for a description of this attribute.
- 272 • **policySets : QName (0..1)** - a list of policy sets. See the [Policy Framework specification](#)
273 [\[POLICY\]](#) for a description of this attribute.

274 2.3.2 SCA Reference element

275 The SCA reference element declares an SCA reference that is made by the Spring application context.
276 When an application context contains one or more SCA reference elements, each of these elements acts
277 as if it were a Spring <bean/> element, offering a target which can satisfy a reference from a <bean/>
278 element within the application context. Each SCA reference element appears as an reference element in
279 the componentType of the Spring implementation and the reference can be configured by the SCA
280 component using that implementation - in particular, the reference can be wired to an appropriate target
281 service.

282 The SCA reference element can also declare other attributes of the SCA reference. In particular, policy
283 can be associated with the reference using the @requires and @policySets attributes.

284 The pseudo-schema for the reference element is:

```
285 <beans xmlns="http://www.springframework.org/schema/beans"
286       xmlns:xs="http://www.w3.org/2001/XMLSchema"
287       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
288       xmlns:sca=
289         "http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810"
290
291     ...
```

```

292     <sca:reference name="xs:NCName"
293                 type="xs:NCName"
294                 default="xs:NCName"?
295                 requires="list of xs:QName"?
296                 policySets="list of xs:QName"?/>
297     ...
298 </beans>
299

```

300 The **reference** element has the following **attributes**:

- 301 • **name : NCName (1..1)** - the name of the reference. The value of the @name attribute of an
302 <sca:reference/> subelement of a <beans/> element MUST be unique amongst the @name
303 attributes of the <reference/> subelements, <property/> subelements and the <bean/>
304 subelements of the <beans/> element. [SPR20003]
- 305 • **type : NCName (1..1)** - the type of the reference, declared as the fully qualified name of a Java
306 class.
- 307 • **default : NCName (0..1)** - the name of a <bean/> element within the application context which
308 provides the reference declared by the sca:reference element if the component using the
309 application context as an implementation does not wire the reference to a target service. The
310 @default attribute of a <reference/> subelement of a <beans/> element MUST have the value of
311 the @name attribute of one of the <bean/> subelements of the <beans/> element. [SPR20004]
- 312 • **requires : QName (0..1)** - a list of policy intents. See the [Policy Framework specification](#)
313 [POLICY] for a description of this attribute.
- 314 • **policySets : QName (0..1)** - a list of policy sets. See the [Policy Framework specification](#)
315 [POLICY] for a description of this attribute.

316 2.3.3 SCA Property element

317 The SCA property element declares an SCA property which can be used by the Spring application
318 context. When an application context contains one or more SCA property elements, each of these
319 elements acts as if it were a Spring <bean/> element, offering a target which can satisfy a reference from a
320 <bean/> element within the application context. Each SCA property element appears as a property
321 element in the componentType of the Spring implementation and the property can be configured by the
322 SCA component using that implementation - the component can provide a value for the property.

323 The pseudo-schema for the property element is:

```

324 <beans xmlns="http://www.springframework.org/schema/beans"
325        xmlns:xs="http://www.w3.org/2001/XMLSchema"
326        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
327        xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca-
328        j/spring/200810"
329
330     ...
331     <sca:property name="xs:NCName"
332                 type="xs:NCName" />
333     ...
334 </beans>
335

```

336 The **property** element has the following **attributes**:

- 337 • **name : NCName (1..1)** - the name of the property. The value of the @name attribute of an
338 <sca:property/> subelement of a <beans/> element MUST be unique amongst the @name
339 attributes of the <property/> subelements, <reference/> subelements and the <bean/>
340 subelements of the <beans/> element. [SPR20005]
- 341 • **type : NCName (1..1)** - the type of the property, declared as the fully qualified name of a Java
342 class.

343

344 2.3.4 Example of a Spring Application Context with SCA Spring Extension 345 Elements

346 The following example shows a Spring application context that exposes one service, SCAService, and
347 explicitly defines an SCA reference, SCAResource. The "bar2" property of bean Y is configured with an
348 SCA property with name "sca-property-name".

```
349     <beans>
350
351         <!-- An explicit reference, which is used by bean "Y" -->
352         <sca:reference name="SCAResource" type="com.xyz.SomeType"/>
353
354         <bean name="X">
355             <property name="foo" ref="Y"/>
356         </bean>
357
358         <bean name="Y">
359             <property name="bar" ref="SCAResource"/>
360             <property name="bar2" ref="sca-property-name"/>
361         </bean>
362
363         <!-- expose an SCA property named "sca-property-name" -->
364         <sca:property name="sca-property-name" type="java.lang.String"/>
365
366         <!-- Expose the bean "X" as an SCA service named "SCAService" -->
367         <sca:service name="SCAService" type="org.xyz.someapp.SomeInterface"
368             target="X"/>
369
370     </beans>
371
```

372 The componentType of the application context is:

```
373     <componentType>
374
375         <service name="SCAService">
376             <interface.java interface="org.xyz.someapp.SomeInterface"/>
377         </service>
378
379         <reference name="SCAResource">
380             <interface.java interface="com.xyz.SomeType"/>
381         </reference>
382
383         <property name="sca-property-name" type="xsd:string"/>
384
385     </componentType>
386
387
```

388 2.3.5 Example of a Spring Application Context without Extension Elements

389 The following example shows a Spring application context that has no SCA extension elements. The
390 application context has one bean, named "X" that is exposed as an SCA service, while a second bean,
391 "Y" has a property that is not satisfied within the application context and has an (introspected) type which
392 is an interface - in SCA terms this unsatisfied property is a reference.

```
393     <beans>
394
395         <bean id="X" class="org.xyz.someapp.SomeClass">
396             <property name="foo" ref="Y"/>
397         </bean>

```



```

398     <bean id="Y" class="org.xyz.someapp.SomeOtherClass">
399         <property name="bar" ref="SCAReference" />
400     </bean>
401
402 </beans>
403

```

404 The componentType of the application context is:

```

405 <componentType>
406     <service name="X">
407         <interface.java interface="org.xyz.someapp.SomeClass" />
408     </service>
409
410     <reference name="SCAReference">
411         <interface.java interface="com.xyz.SomeInterface" />
412     </reference>
413
414 </componentType>
415
416
417

```

418 Here it is assumed that the class "org.xyz.someapp.SomeClass" defines its own interface and that the
419 introspected type of the "bar" property of bean "Y" is "com.xyz.SomeInterface".

420 **2.4 Handling multiple application contexts in SCA runtime**

421 When the <implementation.spring> element's @location attribute specifies the Spring application context
422 file(s) to use via a reference to an archive file or a directory (see the section "Specifying the Spring
423 Implementation Type in an Assembly" for more details) and that location contains more than one Spring
424 application context file, then the SCA runtime has to create a combined application context for the
425 collection of paths identified by the "Spring-Context" header in the MANIFEST.MF file.

426
427 As an example, take the "Spring-Context" header in the MANIFEST.MF file defined as shown below:

```

428 Spring-Context: application-context1.xml; application-context2.xml;
429                 application-context3.xml

```

430 In this case, the SCA runtime has to construct an application context for the set of files identified from the
431 "Spring-Context" header in the MANIFEST.MF file, by configuring the individual application contexts in a
432 hierarchy such that a child application context can see beans defined in a parent, but not vice-versa.

433 In multiple application context scenario, each individual application context definition file identified from
434 the "Spring-Context" header in the MANIFEST.MF file, can have its own SCA services, references and
435 properties defined either implicitly or explicitly.

436 Spring supports the loading of multiple application contexts through other mechanisms. For example,
437 application contexts can be loaded in a parent/child hierarchy using the Spring
438 ClassPathXmlApplicationContext:

```

439
440 <beans>
441     <bean name="bean1" class="....." />
442     <bean name="bean2" class="....." />
443     <bean
444         class="org.springframework.context.support.ClassPathXmlApplicationContext">
445         <constructor-arg>
446             <list>

```

```
447         <value>context1.xml</value>
448         <value>context2.xml</value>
449         <value>context3.xml</value>
450     </list>
451 </constructor-arg>
452 </bean>
453 </beans>
454
```

455 In this case, the 3 contexts context1.xml, context2.xml, context3.xml are loaded by the
456 ClassPathXmlApplicationContext bean as child application contexts. Such application contexts can be
457 loaded and used when the parent context is used as an SCA component implementation, but these
458 application contexts do not contribute to the componentType of the Spring implementation (and they are
459 not introspected by the SCA Spring runtime).

460 In multiple application context scenario, the SCA runtime MUST raise an error when multiple
461 <sca:service> elements are identified with the same name amongst the set of application context files
462 identified from the "Spring-Context" header in the MANIFEST.MF file. [SPR20007]

463 Spring supports duplicate bean definitions for multiple application context scenarios. For example, a bean
464 with the same id or name can be defined in multiple application contexts and in such cases Spring
465 overrides the older bean definition with the later bean definition. When no <sca:service/> element is
466 present in any of the application context file identified from the collection of application context paths
467 identified by the "Spring-Context" header in the MANIFEST.MF file, then the SCA runtime MUST use
468 implicit service determination only for the later bean definition. [SPR20008]

469 In multiple application context scenario, the SCA runtime MUST determine the componentType by
470 applying the rules defined in the section "Component Type of a Spring Application Context" to the
471 combined application context and not to the individual application context files. [SPR20009]

472 For example, when at least one <sca:service/> element is present in any one of the application context
473 file identified from the collection of paths identified by the "Spring-Context" header in the MANIFEST.MF
474 file, then no implicit service determination is used for any of the application contexts and only services
475 explicitly declared with <sca:service/> elements appear in the componentType of the Spring
476 implementation.

477

3 Component Type of a Spring Application Context

478 An SCA runtime MUST introspect the componentType of an implementation.spring application context
479 following the rules defined in the section "Component Type of a Spring Application Context". [SPR30001]

480 The introspected component type MUST be a compliant SCA Java XML Document as defined in section
481 12.1 of [JAVA-CAA]. [SPR30003] The introspected component type uses <interface.java> as the
482 interface type for the introspected services and references. Section 12.1 of [JAVA-CAA] requires
483 compliance with the all requirements of <interface.java>.

484 The component type of a Spring Application Context is introspected from the application context as
485 follows:

486

487 A <service/> element exists for each <sca:service/> element in the application context, where:

- 488 • @name attribute is the value of the @name attribute of the sca:service element
- 489 • @requires attribute is omitted unless the <sca:service/> element has a @requires attribute, in
490 which case the @requires attribute is present with its value equal to the value of the @requires
491 attribute of the <sca:service/> element
- 492 • @policySets attribute is omitted unless the <sca:service/> element has a @policySets attribute,
493 in which case the @policySets attribute is present with its value equal to the value of the
494 @policySets attribute of the <sca:service/> element
- 495 • interface.java child element is present with the @interface attribute set to the fully qualified name
496 of the interface class identified by the @type attribute of the sca:service element. If the @type
497 attribute is not present on the <sca:service/> element, then the interface.java element has its
498 @interface attribute set to the fully qualified name of the Java class of the spring <bean/>
499 element identified by the @target attribute of the <sca:service/> element.
- 500 • binding child element is omitted
- 501 • callback child element is omitted

502

503 If there are no <sca:service/> elements in the application context, one <service/> element exists for each
504 service implemented by each top-level <bean/> element in the application context **except** for bean
505 elements where any of the following apply:

- 506 • <bean/> elements @class attribute is absent
- 507 • <bean/> elements @abstract attribute value is set to "true"
- 508 • <bean/> elements @factory-bean attribute value is set
- 509 • <bean/> elements @factory-method attribute value is set
- 510 • <bean/> elements @parent attribute value is set to reference another bean in the application
511 context
- 512 • <bean/> elements @class attribute value is set to reference the native spring binary classes
513 starting with "org.springframework"

514 where each <service/> element has the following characteristics:

- 515 • @name attribute value is the value of the @id attribute of the <bean/> element if present,
516 otherwise it is the first name from the value of @name attribute of the <bean/> element
- 517 • @requires attribute is omitted
- 518 • @policySets attribute is omitted
- 519 • interface.java child element is present with the @interface attribute set to the fully qualified name
520 of the interface class introspected from the bean class declared in the @class attribute of the

521 <bean/> element, as follows:
522 - if the bean class implements zero Java interfaces annotated with @Remotable, the interface
523 class is the bean class itself
524 - if the bean class implements exactly one Java interface that is annotated with @Remotable, the
525 interface class is the Java interface class which is annotated with @Remotable
526 - where there are no explicit <sca:service/> elements in the application context, if a bean class
527 implements two or more Java interfaces which are annotated with @Remotable, this is not
528 allowed and the SCA runtime MUST raise an error. [SPR30005]

- 529 • binding child element is omitted
- 530 • callback child element is omitted

531

532 Note that as described in the SCA Assembly Model specification

533 **[SCA-ASSEMBLY]** the @name attribute has to be unique amongst all <service/> elements in the
534 componentType.

535 Where a Spring Bean implementation class implements more than one interface, the Bean can be
536 exposed as either a single service or as multiple services, through the use of explicit <sca:service/>
537 elements, where each <sca:service/> element references the same <bean/> element but where the
538 @type attribute uses only one of the interfaces provided by the bean.

539 Where there are no <sca:service/> elements, the bean is exposed as a single service with an interface
540 that is the defined either by the bean class itself, or is defined by a single @Remotable interface
541 implemented by the bean class. It is not premitted for the bean class to implement two or more
542 @Remotable interfaces in this case - this can only be done with the use of explicit <sca:service/>
543 elements.

544 Note that a <bean/> element nested within another <bean/> element (an inner bean) is never exposed
545 directly as an SCA service.

546

547 A <reference/> element exists for each <sca:reference/> element in the application context, where:

- 548 • @name attribute is the value of the @name attribute of the sca:reference element
- 549 • @autowire attribute is omitted
- 550 • @wiredByImpl attribute is omitted
- 551 • @target attribute is omitted
- 552 • @multiplicity attribute is set to (1..1) unless the <sca:reference/> element has the @default
553 attribute present in which case it is set to (0..1)
- 554 • @requires attribute is omitted unless the <sca:reference/> element has a @requires attribute, in
555 which case the @requires attribute is present with its value equal to the value of the @requires
556 attribute of the <sca:reference/> element
- 557 • @policySets attribute is omitted unless the <sca:reference/> element has a @policySets
558 attribute, in which case the @policySets attribute is present with its value equal to the value of the
559 @policySets attribute of the <sca:reference/> element
- 560 • interface.java child element is present, with the interface attribute set to the fully qualified name of
561 the interface class identified by the @type attribute of the <sca:reference/> element
- 562 • binding child element is omitted
- 563 • callback child element is omitted

564

565 A <property/> element exists for each <sca:property/> element in the application context, where:

- 566 • @name attribute is the value of the @name attribute of the <sca:property/> element
- 567 • @value attribute is omitted

- 568 • @type attribute is set to the XML type implied by the JAXB mapping of the Java class identified
- 569 by the @type attribute of the <sca:property/> element
- 570 • @element attribute is omitted
- 571 • @many attribute is set to "false"
- 572 • @mustSupply attribute is set to "true"

573

574 IF there are no <sca:reference/> elements AND no <sca:property> elements in the application context,
575 then references and properties are defined by the bean references in the application context which are
576 not found in the application context as follows:

577

578 A <reference/> element exists for each unique bean reference in the application context to a bean which
579 is not found in the application context and where the bean reference refers to a Java interface class:

- 580 • @name attribute is the value of the @ref attribute of the <property/> or <constructor-arg/>
- 581 element that makes the reference, or the reference name derived from the subelements of the
- 582 <property/> or <constructor-arg/> element (eg. @bean attribute of a <ref/> subelement)
- 583 • @autowire attribute is omitted
- 584 • @wiredByImple attribute is omitted
- 585 • @target attribute is omitted
- 586 • @multiplicity attribute is set to (1..1)
- 587 • @requires attribute is omitted
- 588 • @policySets attribute is omitted
- 589 • interface.java child element is present, with the interface attribute set as follows:
 - 590 1. if only one bean refers to the bean reference, then the interface attribute is set to the fully
 - 591 qualified name of the interface class identified by the bean reference
 - 592 2. if two or more beans refer to the bean reference, each bean reference identifies an
 - 593 interface class. Each interface class in the collection of interface classes has to be either
 - 594 the same as, or an ancestor of, or a descendent of, every other interface class in that
 - 595 collection. **If this condition does not hold true then the SCA runtime MUST raise an error.**
 - 596 **[SPR30002]** . The interface attribute is set to the fully qualified name of the interface
 - 597 class which has the highest depth in the inheritance tree in the set of interface classes -
 - 598 i.e. it is set to the most specific subclass amongst all the interface classes identified by
 - 599 the bean references.

600 For example, if two bean A and B refer to a bean reference C, and the interface class

601 identified by bean A for reference C is I1, and that of bean B is I2, and if I2 is a subclass

602 of I1, then the interface attribute value for the introspected implicit reference is set to I2.

- 603 • binding child element is omitted
- 604 • callback child element is omitted

605

606 A <property/> element exists for each unique bean reference in the application context to a bean which is
607 not found in the application context and where the bean reference does not refer to a Java interface
608 class:

- 609 • @name attribute is the value of the @ref attribute of the <property/> or <constructor-arg/>
- 610 element that makes the reference, or the reference name derived from the subelements of the
- 611 <property/> or <constructor-arg/> element (eg. @bean attribute of a <ref/> subelement)
- 612 • @value attribute is omitted
- 613 • @type attribute is set to the XML type implied by the JAXB mapping of the Java class identified
- 614 by the bean reference

- 615 • @element attribute is omitted
- 616 • @many attribute is set to "false"
- 617 • @mustSupply attribute is set to "true"

618

619 The Spring Component Implementation type does not support the use of Component Type side files,
620 as defined in the SCA Assembly Model specification

621 **[SCA-ASSEMBLY]**, so that the effective componentType of a Spring Application Context is determined
622 completely by introspection of the Spring Application Context.

623 It is beyond the scope of this specification to define the interpretation of the annotations specified in
624 the SCA Common Annotations and API Specification

625 **[JAVA-CAA]**, , except for those annotations that are required to be supported by the SCA Java interface.

626 The SCA runtime **MUST** support the SCA annotations which are applicable to an interface class which is
627 referenced by an <interface.java/> element in the introspected component type of a Spring application
628 context. **[SPR30004]**

629 Other than the annotations which apply to a Java interface referenced from an <interface.java/> element
630 in the component type, an implementation can ignore SCA annotations that are present in classes used
631 by the application context.

632 **3.1 Introspecting the Type Implied by a Spring Bean Reference**

633 In the case where a reference or a property in the component type is derived by introspection of bean
634 references, the type of the reference or property is determined by introspection of the related property
635 setter method or constructor method of the Bean which is the source of the reference.

636 In some cases, the type introspected by this process could be a generic type - for example a List<?>. In
637 such cases, the formal type of the reference becomes Object. This will be interpreted as an SCA
638 property with a Java type of Object, which maps to an XML type of <any/>.

4 Specifying the Spring Implementation Type in an Assembly

639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685

The following pseudo-schema defines the implementation element schema used for the Spring implementation type:.

```
<implementation.spring location="xs:anyURI"
    requires="list of xs:QName"?
    policySets="list of xs:QName"?/>
```

The implementation.spring element has the following attributes:

location : anyURI (1..1) – a URI pointing to the location of the Spring application context to use as the implementation.

The location URI can either be an absolute URI or it can be a relative URI. In the case where the location URI is a relative URI, the URI MUST be taken as being relative to the base of the contribution which contains the composite containing the <implementation.spring/> element. [SPR40009]

The implementation.spring @location attribute URI value MUST point to one of the following:

- a) a Spring application context file
- b) a Java archive file (JAR)
- c) a directory

[SPR40001]

If the implementation.spring @location URI identifies a Spring application context file, it MUST be used as the Spring application context. [SPR40002]

If the implementation.spring @location URI identifies a JAR archive file, then the file META-INF/MANIFEST.MF MUST be read from the archive. [SPR40003]

If the implementation.spring @location URI identifies a directory, then the file META-INF/MANIFEST.MF underneath that directory MUST be read from the directory. [SPR40004]

If the MANIFEST.MF file contains a header "Spring-Context" of the format:

```
Spring-Context ::= path ( ';' path )*
```

where path is a relative path with respect to the @location URI, then each path specified in the header MUST identify a Spring application context configuration file. [SPR40008]

If present, all the Spring application context configuration files identified by the "Spring-Context" header in the MANIFEST.MF file MUST be collectively used to build the Spring application context for implementation.spring element. [SPR40005]

If there is no MANIFEST.MF file or if there is no Spring-Context header within the MANIFEST.MF file, the Spring application context MUST be built using all the *.xml files in the META-INF/spring subdirectory within the JAR identified by the @location URI or underneath the directory specified by the @location URI. [SPR40006]

- **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification \[POLICY\]](#) for a description of this attribute.
- **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification \[POLICY\]](#) for a description of this attribute.

686 The <implementation.spring> element MUST conform to the schema defined in sca-implementation-
687 spring.xsd. [SPR40007]
688

689 5 Conformance

690 The XML schema pointed to by the RDDDL document at the namespace URI, defined by this specification,
691 are considered to be authoritative and take precedence over the XML schema defined in the appendix of
692 this document.

693 There are three categories of artifacts that this specification defines conformance for: SCA Spring
694 Component Implementation Composite Document, SCA Spring Application Context Document and SCA
695 Runtime.

696 5.1 SCA Spring Component Implementation Composite Document

697 An SCA Spring Component Implementation Composite Document is an SCA Composite Document, as
698 defined by the SCA Assembly Model Specification Section 13.1 [ASSEMBLY], that uses the
699 <implementation.spring> element. Such an SCA Spring Component Implementation Composite
700 Document MUST be a conformant SCA Composite Document, as defined by [ASSEMBLY], and MUST
701 comply with additional constraints on the document content as defined in Appendix B.

702 5.2 SCA Spring Application Context Document

703 An SCA Spring Application Context Document is a Spring Framework Application Context Document, as
704 defined by the Spring Framework Specification [SPRING], that uses the SCA Spring extensions defined in
705 Section 2. Such an SCA Spring Application Context Document MUST be a conformant Spring Framework
706 Application Context Document, as defined by [SPRING], and MUST comply with the requirements
707 specified in Section 2 of this specification.

708 5.3 SCA Runtime

709 An implementation that claims to conform to this specification MUST meet the following conditions:

- 710 1. The implementation MUST meet all the conformance requirements defined by the SCA Assembly
711 Model Specification [ASSEMBLY].
- 712 2. The implementation MUST reject an SCA Spring Component Implementation Composite
713 Document that does not conform to the sca-implementation-spring.xsd schema.
- 714 3. The implementation MUST reject an SCA Spring Application Context Document that does not
715 conform to the sca-spring-extensions.xsd schema.
- 716 4. The implementation MUST comply with all statements related to an SCA Runtime, specified in
717 'Appendix B: Conformance Items' of this specification, notably all mandatory statements have to
718 be implemented.

719

720

A. XML Schemas

721

A.1 sca-implementation-spring.xsd

722

```
<?xml version="1.0" encoding="UTF-8"?>
```

723

```
<!-- Copyright(C) OASIS(R) 2005,2011. All Rights Reserved.
```

724

```
    OASIS trademark, IPR and other policies apply. -->
```

725

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
```

726

```
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
```

727

```
  elementFormDefault="qualified"
```

728

```
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912">
```

729

```
  <include schemaLocation="sca-core-1.1-cd06.xsd"/>
```

730

```
  <element name="implementation.spring" type="sca:SpringImplementation"
```

731

```
    substitutionGroup="sca:implementation"/>
```

732

```
  <complexType name="SpringImplementation">
```

733

```
    <complexContent>
```

734

```
      <extension base="sca:Implementation">
```

735

```
        <sequence>
```

736

```
          <any namespace="##other" processContents="lax" minOccurs="0"
```

737

```
            maxOccurs="unbounded"/>
```

738

```
        </sequence>
```

739

```
        <attribute name="location" type="anyURI" use="required"/>
```

740

```
      </extension>
```

741

```
    </complexContent>
```

742

```
  </complexType>
```

743

```
</schema>
```

744

745

746

A.2 SCA Spring Extension schema

747

- sca-spring-extensions.xsd

748

```
<?xml version="1.0" encoding="UTF-8"?>
```

749

```
<!-- Copyright(C) OASIS(R) 2005,2011. All Rights Reserved.
```

750

```
    OASIS trademark, IPR and other policies apply. -->
```

751

```
<xsd:schema
```

752

```
  xmlns="http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810"
```

753

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

754

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

755

```
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
```

756

```
  xsi:schemaLocation="
```

757

```
    http://docs.oasis-open.org/ns/opencsa/sca/200912
```

758

```
    http://docs.oasis-open.org/opencsa/sca-assembly/sca-core-1.1-cd06.xsd"
```

759

```
  attributeFormDefault="unqualified"
```

760

```
  elementFormDefault="qualified"
```

761

```
  targetNamespace="
```

762

```
    "http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810">
```

763

```
  <xsd:element name="reference">
```

764

```
    <xsd:complexType>
```

765

```
      <any namespace="##other" processContents="lax"
```

766

```
        minOccurs="0" maxOccurs="unbounded"/>
```

767

```
      <xsd:attribute name="name" type="xsd:NCName"
```

768

```
        use="required"/>
```

769

```
      <xsd:attribute name="type" type="xsd:NCName"
```

770


```

771         use="required"/>
772     <xsd:attribute name="default" type="xsd:NCName"
773         use="optional"/>
774     <xsd:attribute name="requires" type="sca:listOfQNames"
775         use="optional"/>
776     <xsd:attribute name="policySets" type="sca:listOfQNames"
777         use="optional"/>
778     <xsd:anyAttribute namespace="##other" processContents="lax"
779         use="optional"/>
780 </xsd:complexType>
781 </xsd:element>
782
783 <xsd:element name="property">
784     <xsd:complexType>
785         <any namespace="##other" processContents="lax"
786             minOccurs="0" maxOccurs="unbounded"/>
787         <xsd:attribute name="name" type="xsd:NCName"
788             use="required"/>
789         <xsd:attribute name="type" type="xsd:NCName"
790             use="required"/>
791         <xsd:anyAttribute namespace="##other" processContents="lax"
792             use="optional"/>
793     </xsd:complexType>
794 </xsd:element>
795
796 <xsd:element name="service">
797     <xsd:complexType>
798         <any namespace="##other" processContents="lax"
799             minOccurs="0" maxOccurs="unbounded"/>
800         <xsd:attribute name="name" type="xsd:NCName"
801             use="required"/>
802         <xsd:attribute name="type" type="xsd:NCName"
803             use="optional"/>
804         <xsd:attribute name="target" type="xsd:NCName"
805             use="required"/>
806         <xsd:attribute name="requires" type="sca:listOfQNames"
807             use="optional"/>
808         <xsd:attribute name="policySets" type="sca:listOfQNames"
809             use="optional"/>
810         <xsd:anyAttribute namespace="##other" processContents="lax"
811             use="optional"/>
812     </xsd:complexType>
813 </xsd:element>
814
815 </xsd:schema>

```

B. Conformance Items

Conformance ID	Description
[SPR20001]	The value of the @name attribute of an <sca:service/> subelement of a <beans/> element MUST be unique amongst the <service/> subelements of the <beans/> element.
[SPR20002]	The @target attribute of a <service/> subelement of a <beans/> element MUST have the value of the @name attribute of one of the <bean/> subelements of the <beans/> element.
[SPR20003]	The value of the @name attribute of an <sca:reference/> subelement of a <beans/> element MUST be unique amongst the @name attributes of the <reference/> subelements, <property/> subelements and the <bean/> subelements of the <beans/> element.
[SPR20004]	The @default attribute of a <reference/> subelement of a <beans/> element MUST have the value of the @name attribute of one of the <bean/> subelements of the <beans/> element.
[SPR20005]	The value of the @name attribute of an <sca:property/> subelement of a <beans/> element MUST be unique amongst the @name attributes of the <property/> subelements, <reference/> subelements and the <bean/> subelements of the <beans/> element.
[SPR20006]	SCA extension elements within a Spring application context MUST conform to the SCA Spring Extension schema declared in sca-spring-extensions.xsd.
[SPR20007]	In multiple application context scenario, the SCA runtime MUST raise an error when multiple <sca:service> elements are identified with the same name amongst the set of application context files identified from the "Spring-Context" header in the MANIFEST.MF file.
[SPR20008]	When no <sca:service/> element is present in any of the application context file identified from the collection of application context paths identified by the "Spring-Context" header in the MANIFEST.MF file, then the SCA runtime MUST use implicit service determination only for the later bean definition.
[SPR20009]	In multiple application context scenario, the SCA runtime MUST determine the componentType by applying the rules defined in the section "Component Type of a Spring Application Context" to the combined application context and not to the individual application context files.
[SPR30001]	An SCA runtime MUST introspect the componentType of an implementation.spring application context following the rules defined in the section "Component Type of a Spring Application Context".
[SPR30002]	If this condition does not hold true then the SCA runtime MUST raise an error.
[SPR30003]	The introspected component type MUST be a compliant SCA Java XML Document as defined in section 12.1 of [JAVA-CAA].
[SPR30004]	The SCA runtime MUST support the SCA annotations which are applicable to an interface class which is referenced by an <interface.java/> element in the introspected component type of a Spring application context.

[SPR30005]	where there are no explicit <sca:service/> elements in the application context, if a bean class implements two or more Java interfaces which are annotated with @Remotable, this is not allowed and the SCA runtime MUST raise an error.
[SPR40001]	The implementation.spring @location attribute URI value MUST point to one of the following: a) a Spring application context file b) a Java archive file (JAR) c) a directory
[SPR40002]	If the implementation.spring @location URI identifies a Spring application context file, it MUST be used as the Spring application context.
[SPR40003]	If the implementation.spring @location URI identifies a JAR archive file, then the file META-INF/MANIFEST.MF MUST be read from the archive.
[SPR40004]	If the implementation.spring @location URI identifies a directory, then the file META-INF/MANIFEST.MF underneath that directory MUST be read from the directory.
[SPR40005]	If present, all the Spring application context configuration files identified by the "Spring-Context" header in the MANIFEST.MF file MUST be collectively used to build the Spring application context for implementation.spring element.
[SPR40006]	If there is no MANIFEST.MF file or if there is no Spring-Context header within the MANIFEST.MF file, the Spring application context MUST be built using all the *.xml files in the META-INF/spring subdirectory within the JAR identified by the @location URI or underneath the directory specified by the @location URI.
[SPR40007]	The <implementation.spring> element MUST conform to the schema defined in sca-implementation-spring.xsd.
[SPR40008]	If the MANIFEST.MF file contains a header "Spring-Context" of the format: Spring-Context ::= path (';' path)* where path is a relative path with respect to the @location URI, then each path specified in the header MUST identify a Spring application context configuration file.
[SPR40009]	The location URI can either be an absolute URI or it can be a relative URI. In the case where the location URI is a relative URI, the URI MUST be taken as being relative to the base of the contribution which contains the composite containing the <implementation.spring/> element.

818 **C. Acknowledgements**

819 The following individuals have participated in the creation of this specification and are gratefully
820 acknowledged:

821 **Participants:**

822

Participant Name	Affiliation
Bryan Aupperle	IBM
Ron Barack	SAP AG*
Mirza Begg	Individual
Michael Beisiegel	IBM
Henning Blohm	SAP AG*
David Booz	IBM
Martin Chapman	Oracle Corporation
Graham Charters	IBM
Shih-Chang Chen	Oracle Corporation
Chris Cheng	Primeton Technologies, Inc.
Vamsavardhana Reddy Chillakuru	IBM
Roberto Chinnici	Sun Microsystems
Pyounguk Cho	Oracle Corporation
Eric Clairambault	IBM
Mark Combella	Avaya, Inc.
Jean-Sebastien Delfino	IBM
Derek Dougans	Individual
Mike Edwards	IBM
Ant Elder	IBM
Raymond Feng	IBM
James Hao	Primeton Technologies, Inc.
Bo Ji	Primeton Technologies, Inc.
Uday Joshi	Oracle Corporation
Anish Karmarkar	Oracle Corporation
Khanderao Kand	Oracle Corporation
Michael Keith	Oracle Corporation
Rainer Kerth	SAP AG*
Meeraj Kunnumpurath	Individual
Simon Laws	IBM
Yang Lei	IBM
Mark Little	Red Hat
Ashok Malhotra	Oracle Corporation
Jim Marino	Individual
Jeff Mischkinsky	Oracle Corporation
Sriram Narasimhan	TIBCO Software Inc.
Simon Nash	Individual
Sanjay Patil	SAP AG*
Plamen Pavlov	SAP AG*
Peter Peshev	SAP AG*

Ramkumar Ramalingam
Luciano Resende
Michael Rowley
Vladimir Savchenko
Pradeep Simha
Raghav Srinivasan
Scott Vorthmann
Feng Wang

Paul Yang
Yuan Yi

IBM
IBM
Active Endpoints, Inc.
SAP AG*
TIBCO Software Inc.
Oracle Corporation
TIBCO Software Inc.
Primeton Technologies, Inc.
Changfeng Open Standards
Platform Software
Primeton Technologies, Inc.

824

D. Revision History

825 [optional; should not be included in OASIS Standards]

826

Revision	Date	Editor	Changes Made
1	2007-09-26	Anish Karmarkar	Applied the OASIS template + related changes to the Submission
WD01	2008-11-24	Mike Edwards	Editorial cleanup Issue 64 resolution applied Issue 57 resolution applied
WD02	2009-07-20	Mike Edwards	Issue 164 resolution applied Added Appendix B - Conformance Items Issue 58 resolution applied (new Section 3) Issue 92 resolution applied - Section 3 Issue 59 resolution applied - Section 3
WD02 + Issue106	2009-08-06	Mike Edwards	Issue 106 (RFC2119) - added Section 4 - added Appendix A1 - added Appendix B
WD03	2009-08-07	Mike Edwards	All changes accepted.
WD04	2009-08-14	Mike Edwards	Issue 63 applied - Section 2 All changes accepted
WD05	2010-08-06	Anish Karmarkar	Issue 63 fully applied (few changes from the resolution were missing) Issue 149 resolution applied. Issue 166 resolution applied. Issue 167 resolution applied. Issue 173 & 175 resolution applied. Issue 150 resolution applied.
WD06	2011-02-23	Anish Karmarkar	Issue 182 resolution applied Issue 225 resolution applied Issue 229 resolution applied Ed fixes
WD07	2011-03-29	Mike Edwards	Issue 230 resolution applied Issue 237 resolution applied Formatting updates & editorial fixes. Added Acknowledgements entries

WD08	2011-05-02	Anish Karmarkar	Issue 228 resolution applied Added a ref to testcases doc and a new section that talks about it
------	------------	-----------------	--

827

828

829