



Service Component Architecture POJO Component Implementation Specification Version 1.1

Committee Draft 01/Public Review Draft 01 rev⁴

Deleted: 3

18th January 2010

Specification URIs:

This Version:

<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-cd01.html>
<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-cd01.doc>
<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-cd01.pdf> (Authoritative)

Previous Version:

Latest Version:

<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec.html>
<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec.doc>
<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec.pdf> (Authoritative)

Latest Approved Version:

Technical Committee:

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

Chair(s):

David Booz, IBM
Mark Combella, Avaya

Editor(s):

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

Related work:

This specification replaces or supersedes:

- Service Component Architecture Java Component Implementation Specification Version 1.00, 15 February 2007

This specification is related to:

- Service Component Architecture Assembly Model Specification Version 1.1
- Service Component Architecture Policy Framework Specification Version 1.1
- Service Component Architecture SCA-J Common Annotations and APIs Specification Version 1.1

Declared XML Namespace(s):

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

Abstract:

This specification extends the SCA Assembly Model by defining how a Java class provides an implementation of an SCA component, including its various attributes such as services, references, and properties and how that class is used in SCA as a component implementation type. It requires all the annotations and APIs as defined by the SCA-J Common Annotations and APIs specification.

This specification also details the use of metadata and the Java API defined in the context of a Java class used as a component implementation type.

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" or "Latest Approved 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>).

The non-normative errata page for this specification is located at <http://www.oasis-open.org/committees/sca-j/>.

Notices

Copyright © OASIS® 2005, 2010. 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 names "OASIS", [insert specific trademarked names and abbreviations here] are trademarks 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	6
1.1	Terminology	6
1.2	Normative References	6
2	Service	7
2.1	Use of @Service	7
2.2	Local and Remotable Services	9
2.3	Introspecting Services Offered by a Java Implementation	11
2.4	Non-Blocking Service Operations	11
2.5	Callback Services	11
3	References	12
3.1	Reference Injection	12
3.2	Dynamic Reference Access	12
4	Properties	13
4.1	Property Injection	13
4.2	Dynamic Property Access	14
5	Implementation Instance Creation	15
6	Implementation Scopes and Lifecycle Callbacks	17
7	Accessing a Callback Service	18
8	Component Type of a Java Implementation	19
8.1	Component Type of an Implementation with no @Service, @Reference or @Property Annotations	20
8.2	Impact of JAX-WS Annotations on ComponentType	22
8.2.1	@WebService	22
8.2.2	@WebMethod	22
8.2.3	@WebParam	22
8.2.4	@WebResult	22
8.2.5	@SOAPBinding	23
8.2.6	@WebFault	23
8.2.7	@WebServiceProvider	23
8.2.8	Web Service Binding	23
8.3	Component Type Introspection Examples	23
8.4	Java Implementation with Conflicting Setter Methods	24
9	Specifying the Java Implementation Type in an Assembly	26
10	Java Packaging and Deployment Model	27
10.1	Contribution Metadata Extensions	27
10.2	Java Artifact Resolution	29
10.3	Class Loader Model	29
11	Conformance	30
11.1	SCA Java Component Implementation Composite Document	30
11.2	SCA Java Component Implementation Contribution Document	30
11.3	SCA Runtime	30
A.	XML Schemas	31
A.1	sca-contribution-java.xsd	31

A.2 sca-implementation-java.xsd	31
B. Conformance Items.....	33
C. Acknowledgements.....	35
D. Non-Normative Text.....	37
E. Revision History.....	38

1 Introduction

This specification extends the SCA Assembly Model [ASSEMBLY] by defining how a Java class provides an implementation of an SCA component (including its various attributes such as services, references, and properties) and how that class is used in SCA as a component implementation type.

This specification requires all the annotations and APIs as defined by the SCA-J Common Annotations and APIs specification [JAVACAA]. All annotations and APIs referenced in this document are defined in the former unless otherwise specified. Moreover, the semantics defined in the SCA-J Common Annotations and APIs specification are normative.

In addition, it details the use of metadata and the Java API defined in the SCA-J Common Annotations and APIs Specification [JAVACAA] in the context of a Java class used as a component implementation type

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*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- [ASSEMBLY] SCA Assembly Model Specification Version 1.1, <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.pdf>
- [POLICY] SCA Policy Framework Specification Version 1.1, <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.pdf>
- [JAVACAA] Service Component Architecture SCA-J Common Annotations and APIs Specification Version 1.1, <http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd03.pdf>
- [WSDL] WSDL Specification, WSDL 1.1: <http://www.w3.org/TR/wsdl>
- [OSGi Core] OSGi Service Platform Core Specification, Version 4.0.1 <http://www.osgi.org/download/r4v41/r4.core.pdf>
- [JAVABEANS] JavaBeans 1.01 Specification, <http://java.sun.com/javase/technologies/desktop/javabeans/api/>
- [JAX-WS] JAX-WS 2.1 Specification (JSR-224), <http://www.jcp.org/en/jsr/detail?id=224>
- [WSBINDING] SCA Web Service Binding Specification Version 1.1, <http://docs.oasis-open.org/opencsa/sca-bindings/sca-wsbinding-1.1-spec-cd03.pdf>

45 2 Service

46 A component implementation based on a Java class can provide one or more services.

47 The services provided by a Java-based implementation MUST have an interface defined in one of the
48 following ways:

- 49 • A Java interface
- 50 • A Java class
- 51 • A Java interface generated from a Web Services Description Language [WSDL] (WSDL) portType.

52 [JCI20001]

53 Java implementation classes MUST implement all the operations defined by the service interface.

54 [JCI20002] If the service interface is defined by a Java interface, the Java-based component can either
55 implement that Java interface, or implement all the operations of the interface.

56 Java interfaces generated from WSDL portTypes are remotable, see the WSDL to Java and Java to
57 WSDL section of the SCA-J Common Annotations and APIs Specification [JAVACAA] for details.

58 A Java implementation type can specify the services it provides explicitly through the use of the @Service
59 annotation. In certain cases as defined below, the use of the @Service annotation is not necessary and
60 the services a Java implementation type offers can be inferred from the implementation class itself.

61 2.1 Use of @Service

62 Service interfaces can be specified as a Java interface. A Java class, which is a component
63 implementation, can offer a service by implementing a Java interface specifying the service contract. As a
64 Java class can implement multiple interfaces, some of which might not define SCA services, the
65 @Service annotation can be used to indicate the services provided by the implementation and their
66 corresponding Java interface definitions.

67 **Snippet 2-1, and Error! Reference source not found. are** an example of a Java service interface and a
68 Java implementation which provides a service using that interface:

Deleted: The following

Deleted: is

69 Interface:

```
70 package services.hello;  
71  
72 public interface HelloService {  
73     String hello(String message);  
74 }  
75
```

Deleted: 2-1

Formatted: Caption

76 **Snippet 2-1: Example Java Service Interface**

77

78 Implementation class:

```
79 @Service(HelloService.class)  
80 public class HelloServiceImpl implements HelloService {  
81  
82     public String hello(String message) {  
83         ...  
84     }  
85 }
```

Formatted: Caption

86 **Snippet 2-2: Example Java Component Implementation**

87

88 | The XML representation of the component type for this implementation is shown in [Snippet 2-3](#) for
89 | illustrative purposes. There is no need to author the component type as it is introspected from the Java
90 | class.

Deleted: below

91
92
93
94
95
96
97
98
99

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
  <service name="HelloService">
    <interface.java interface="services.hello.HelloService"/>
  </service>
</componentType>
```

Formatted: Caption, Don't adjust space between Latin and Asian text

100 | [Snippet 2-3: Effective Component Type for Implementation in Snippet 2-2](#)

101

102 | Another possibility is to use the Java implementation class itself to define a service offered by a
103 | component and the interface of the service. In this case, the `@Service` annotation can be used to
104 | explicitly declare the implementation class defines the service offered by the implementation. In this case,
105 | a component will only offer services declared by `@Service`. [Snippet 2-4](#) illustrates this:

Deleted: The following

106
107
108
109
110
111
112
113
114
115
116

```
package services.hello;

@Service(HelloServiceImpl.class)
public class HelloServiceImpl implements AnotherInterface {

    public String hello(String message) {
        ...
    }
    ...
}
```

Formatted: Caption

117 | [Snippet 2-4: Example of Java Class Defining a Service](#)

118

119 | In [Snippet 2-4](#), `HelloServiceImpl` offers one service as defined by the public methods of the
120 | implementation class. The interface `AnotherInterface` in this case does not specify a service offered by
121 | the component. [Snippet 2-5](#) is an XML representation of the introspected component type:

Deleted: the above example

Deleted: The following

122
123
124
125
126
127
128
129

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
  <service name="HelloServiceImpl">
    <interface.java interface="services.hello.HelloServiceImpl"/>
  </service>
</componentType>
```

Formatted: Caption, Don't adjust space between Latin and Asian text

130 | [Snippet 2-5: Effective Component Type for Implementation in Snippet 2-4](#)

131

132 | The `@Service` annotation can be used to specify multiple services offered by an implementation as in
133 | [Snippet 2-6](#):

Deleted: the following example

134
135
136
137
138
139
140

```
@Service(interfaces={HelloService.class, AnotherInterface.class})
public class HelloServiceImpl implements HelloService, AnotherInterface
{
    public String hello(String message) {
        ...
    }
}
```

```
141 }
142 ...
143 }
```

Formatted: Caption

144 [Snippet 2-6: Example of @Service Specifying Multiple Services](#)

145
146 [Snippet 2-7](#) shows the introspected component type for this implementation.

Deleted: The following snippet

```
147 <?xml version="1.0" encoding="UTF-8"?>
148 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
149
150   <service name="HelloService">
151     <interface.java interface="services.hello.HelloService"/>
152   </service>
153   <service name="AnotherService">
154     <interface.java interface="services.hello.AnotherService"/>
155   </service>
156
157 </componentType>
```

Formatted: Caption, Don't adjust space between Latin and Asian text

158 [Snippet 2-7: Effective Component Type for Implementation in Snippet 2-6](#)

159 2.2 Local and Remotable Services

160 A Java interface or implementation class that defines an SCA service can use the @Remotable
161 annotation to declare that the service follows the semantics of remotable services as defined by the SCA
162 Assembly Model Specification [ASSEMBLY]. [Snippet 2-8 and Snippet 2-9](#) demonstrate the use of the
163 @Remotable annotation on a Java interface:

Deleted: The following example

Deleted: s

164 Interface:

```
165 package services.hello;
166
167 @Remotable
168 public interface HelloService {
169     String hello(String message);
170 }
171 }
```

Formatted: Caption

172 [Snippet 2-8: Example Remotable Interface](#)

173
174 Implementation class:

```
175 package services.hello;
176
177 @Service(HelloService.class)
178 public class HelloServiceImpl implements HelloService {
179
180     public String hello(String message) {
181         ...
182     }
183 }
```

Formatted: Caption

184 [Snippet 2-9: Implementation for Remotable Interface](#)

185
186 [Snippet 2-10](#) shows the introspected component type for this implementation.

Deleted: The following snippet

```
187 <?xml version="1.0" encoding="UTF-8"?>
188 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
189   <service name="HelloService">
190     <interface.java interface="services.hello.HelloService"/>
191   </service>
192 </componentType>
```

193 [Snippet 2-10: Effective Component Type for Implementation in Snippet 2-9](#)

Formatted: Caption, Don't adjust space between Latin and Asian text

194
195 The interface specified in the @interface attribute of the <interface.java/> element is implicitly remotable
196 because the Java interface contains @Remotable.

197 If a service is defined by a Java implementation class instead of a Java interface, the @Remotable
198 annotation can be used on the implementation class to indicate that the service is remotable. [Snippet](#)
199 [2-11](#) demonstrates this:

Deleted: The following example

```
200 package services.hello;  
201  
202 @Remotable  
203 @Service(HelloServiceImpl.class)  
204 public class HelloServiceImpl {  
205  
206     public String hello(String message) {  
207         ...  
208     }  
209 }
```

Formatted: Caption

210 [Snippet 2-11: Remotable Interface Defined by a Class](#)

211
212 [Snippet 2-12](#) shows the introspected component type for this implementation.

Deleted: The following snippet

```
213 <?xml version="1.0" encoding="UTF-8"?>  
214 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
215     <service name="HelloServiceImpl">  
216         <interface.java interface="services.hello.HelloServiceImpl"/>  
217     </service>  
218 </componentType>
```

Formatted: Caption, Don't adjust space between Latin and Asian text

219 [Snippet 2-12: Effective Component Type for Implementation in Snippet 2-11](#)

220
221 The interface specified in the @interface attribute of the <interface.java/> element is implicitly remotable
222 because the Java implementation class contains @Remotable.

223 It is also possible to use a Java interface with no @Remotable annotation to define an SCA service with
224 remotable semantics. In this case, the @Remotable annotation is placed on the service implementation
225 class, as shown in [Snippet 2-13](#) and [Snippet 2-14](#):

Deleted: the following example

226 Interface:

```
227 package services.hello;  
228  
229 public interface HelloService {  
230  
231     String hello(String message);  
232 }
```

Formatted: Caption

233 [Snippet 2-13: Interface without @Remotable](#)

234
235 Implementation class:

```
236 package services.hello;  
237  
238 @Remotable  
239 @Service(HelloService.class)  
240 public class HelloServiceImpl implements HelloService {  
241  
242     public String hello(String message) {  
243         ...  
244     }
```

245 | } 

246 | [Snippet 2-14: Interface Made Remotable with @Remotable on Implementation Class](#)

Formatted: Caption

247 |
248 | In this case the introspected component type for the implementation uses the @remotable attribute of the
249 | <interface.java/> element, as shown in [Snippet 2-15](#);

Deleted: the following snippet

```
250 | <?xml version="1.0" encoding="UTF-8"?>  
251 | <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
252 |   <service name="HelloService">  
253 |     <interface.java interface="services.hello.HelloService"  
254 |       remotable="true"/>  
255 |   </service>  
256 | </componentType>
```

257 | [Snippet 2-15: Effective Component Type for Implementation in Snippet 2-14](#)

Formatted: Caption, Don't adjust space between Latin and Asian text

258 |
259 | An SCA service defined by a @Service annotation specifying a Java interface, with no @Remotable
260 | annotation on either the interface or the service implementation class, is inferred to be a local service as
261 | defined by the SCA Assembly Model Specification [ASSEMBLY]. Similarly, an SCA service defined by a
262 | @Service annotation specifying a Java implementation class with no @Remotable annotation is inferred
263 | to be a local service.

264 | An implementation class can provide hints to the SCA runtime about whether it can achieve pass-by-
265 | value semantics without making a copy by using the @AllowsPassByReference annotation.

266 | 2.3 Introspecting Services Offered by a Java Implementation

267 | The services offered by a Java implementation class are determined through introspection, as defined in
268 | the section "[Component Type of a Java Implementation](#)".

269 | If the interfaces of the SCA services are not specified with the @Service annotation on the
270 | implementation class and the implementation class does not contain any @Reference or @Property
271 | annotations, it is assumed that all implemented interfaces that have been annotated as @Remotable are
272 | the service interfaces provided by the component. If an implementation class has only implemented
273 | interfaces that are not annotated with a @Remotable annotation, the class is considered to implement a
274 | single **local** service whose type is defined by the class (note that local services can be typed using either
275 | Java interfaces or classes).

276 | 2.4 Non-Blocking Service Operations

277 | Service operations defined by a Java interface can use the @OneWay annotation to declare that the SCA
278 | runtime needs to honor non-blocking semantics as defined by the SCA Assembly Model Specification
279 | [ASSEMBLY] when a client invokes the service operation.

280 | 2.5 Callback Services

281 | A callback interface can be declared by using the @Callback annotation on the service interface or Java
282 | implementation class as described in the SCA-J Common Annotations and APIs Specification
283 | [JAVACAA]. Alternatively, the @callbackInterface attribute of the <interface.java/> element can be used
284 | to declare a callback interface.

285 3 References

286 A Java implementation class can obtain **service references** either through injection or through the
287 ComponentContext API as defined in the SCA-J Common Annotations and APIs Specification
288 [JAVACAA]. When possible, the preferred mechanism for accessing references is through injection.

289 3.1 Reference Injection

290 A Java implementation type can explicitly specify its references through the use of the @Reference
291 annotation as in [Snippet 3-1](#);

Deleted: the following example

```
292  
293 public class ClientComponentImpl implements Client {  
294     private HelloService service;  
295  
296     @Reference  
297     public void setHelloService(HelloService service) {  
298         this.service = service;  
299     }  
300 }
```

Formatted: Caption

301 [Snippet 3-1: Specifying a Reference](#)

302
303 If @Reference marks a setter method, the SCA runtime provides the appropriate implementation of the
304 service reference contract as specified by the parameter type of the method. This is done by invoking the
305 setter method of an implementation instance of the Java class. When injection occurs is defined by the
306 **scope** of the implementation. However, injection always occurs before the first service method is called.

307 If @Reference marks a field, the SCA runtime provides the appropriate implementation of the service
308 reference contract as specified by the field type. This is done by setting the field on an implementation
309 instance of the Java class. When injection occurs is defined by the scope of the implementation.
310 However, injection always occurs before the first service method is called.

311 If @Reference marks a parameter on a constructor, the SCA runtime provides the appropriate
312 implementation of the service reference contract as specified by the constructor parameter during
313 creation of an implementation instance of the Java class.

314 Except for constructor parameters, references marked with the @Reference annotation can be declared
315 with `required=false`, as defined by the SCA-J Common Annotations and APIs Specification [JAVACAA] -
316 i.e., the reference multiplicity is 0..1 or 0..n, where the implementation is designed to cope with the
317 reference not being wired to a target service.

318 The @Remotable annotation can be used either on the service reference contract or on the reference
319 itself to specify that the service reference contract follows the semantics of remotable services as defined
320 by the SCA Assembly Model Specification [ASSEMBLY]; otherwise, the service reference contract has
321 local semantics.

322 In the case where a Java class contains no @Reference or @Property annotations, references are
323 determined by introspecting the implementation class as described in the section "[ComponentType of an](#)
324 [Implementation with no @Reference or @Property annotations](#)".

325 3.2 Dynamic Reference Access

326 As an alternative to reference injection, service references can be accessed dynamically through the API
327 methods ComponentContext.getService() and ComponentContext.getServiceReference() methods as
328 described in the SCA-J Common Annotations and APIs Specification [JAVACAA].

329 4 Properties

330 4.1 Property Injection

331 Properties can be obtained either through injection or through the ComponentContext API as defined in
332 the SCA-J Common Annotations and APIs Specification [JAVACAA]. When possible, the preferred
333 mechanism for accessing properties is through injection.

334 A Java implementation type can explicitly specify its properties through the use of the @Property
335 annotation as in [Snippet 4-1](#);

Deleted: the following example

```
336  
337 public class ClientComponentImpl implements Client {  
338     private int maxRetries;  
339  
340     @Property  
341     public void setMaxRetries(int maxRetries) {  
342         this.maxRetries = maxRetries;  
343     }  
344 }
```

Formatted: Caption

345 [Snippet 4-1: Specifying a Property](#)

346
347 If the @Property annotation marks a setter method, the SCA runtime provides the appropriate property
348 value by invoking the setter method of an implementation instance of the Java class. When injection
349 occurs is defined by the scope of the implementation. However, injection always occurs before the first
350 service method is called.

351 If the @Property annotation marks a field, the SCA runtime provides the appropriate property value by
352 setting the value of the field of an implementation instance of the Java class. When injection occurs is
353 defined by the scope of the implementation. However, injection always occurs before the first service
354 method is called.

355 If the @Property annotation marks a parameter on a constructor, the SCA runtime provides the
356 appropriate property value during creation of an implementation instance of the Java class.

357 Except for constructor parameters, properties marked with the @Property annotation can be declared
358 with required=false as defined by the SCA-J Common Annotations and APIs Specification [JAVACAA],
359 i.e., the property mustSupply attribute is false and where the implementation is designed to cope with the
360 component configuration not supplying a value for the property.

361 In the case where a Java class contains no @Reference or @Property annotations, properties are
362 determined by introspecting the implementation class as described in the section "[ComponentType of an
363 Implementation with no @Reference or @Property annotations](#)".

364 For an unannotated field or setter method that is introspected as a property and where the Java type of
365 the field or setter method is a JAXB [JAXB] annotated class, the SCA runtime MUST convert a property
366 value specified by an SCA component definition into an instance of the property's Java type as defined by
367 the XML to Java mapping in the JAXB specification [JAXB] with XML schema validation enabled.
368 [JCI40001]

369 For an unannotated field or setter method that is introspected as a property and where the Java type of
370 the field or setter method in not a JAXB [JAXB] annotated class, the SCA runtime can use any XML to
371 Java mapping when converting property values into instances of the Java type.

372

4.2 Dynamic Property Access

Deleted: 1

373
374
375

As an alternative to property injection, properties can also be accessed dynamically through the `ComponentContext.getProperty()` method as described in the SCA-J Common Annotations and APIs Specification [JAVACAA].

376 5 Implementation Instance Creation

377 A Java implementation class MUST provide a public or protected constructor that can be used by the
378 SCA runtime to create the implementation instance. [JCI50001] The constructor can contain parameters;
379 in the presence of such parameters, the SCA container passes the applicable property or reference
380 values when invoking the constructor. Any property or reference values not supplied in this manner are
381 set into the field or are passed to the setter method associated with the property or reference before any
382 service method is invoked.

383 The constructor to use for the creation of an implementation instance MUST be selected by the SCA
384 runtime using the sequence:

- 385 1. A declared constructor annotated with a @Constructor annotation.
- 386 2. A declared constructor, all of whose parameters are annotated with either @Property or
387 @Reference.
- 388 3. A no-argument constructor.

Formatted: Bullets and Numbering

389 [JCI50004]

390 The @Constructor annotation MUST NOT appear on more than one constructor. [JCI50002]

391 In the absence of an @Constructor annotation, there MUST NOT be more than one constructor that has
392 a non-empty parameter list with all parameters annotated with either @Property or @Reference.

393 [JCI50005]

394 The property or reference associated with each parameter of a constructor is identified through the
395 presence of a @Property or @Reference annotation on the parameter declaration.

396 The construction and initialization of component implementation instances is described as part of the SCA
397 component implementation lifecycle in the SCA-J Common Annotations and APIs specification
398 [JAVACAA].

399 Snippet 5-1 shows examples of legal Java component constructor declarations:

Deleted: ¶

Deleted: The following are

Formatted: Normal, Don't adjust space between Latin and Asian text

```
400 /** Constructor declared using @Constructor annotation */
401 public class Impl1 {
402     private String someProperty;
403     @Constructor
404     public Impl1( @Property("someProperty") String propval ) {...}
405 }
406
407 /** Declared constructor unambiguously identifying all Property
408 * and Reference values */
409 public class Impl2 {
410     private String someProperty;
411     private SomeService someReference;
412     public Impl2( @Property("someProperty") String a,
413                 @Reference("someReference") SomeService b )
414     {...}
415 }
416
417 /** Declared constructor unambiguously identifying all Property
418 * and Reference values plus an additional Property injected
419 * via a setter method */
420 public class Impl3 {
421     private String someProperty;
422     private String anotherProperty;
423     private SomeService someReference;
424     public Impl3( @Property("someProperty") String a,
425                 @Reference("someReference") SomeService b)
426     {...}
427     @Property
```

```
428     public void setAnotherProperty( String anotherProperty ) {...}
429 }
430
431 /** No-arg constructor */
432 public class Impl4 {
433     @Property
434     public String someProperty;
435     @Reference
436     public SomeService someReference;
437     public Impl4() {...}
438 }
439
440 /** Unannotated implementation with no-arg constructor */
441 public class Impl5 {
442     public String someProperty;
443     public SomeService someReference;
444     public Impl5() {...}
445 }
```

446 [Snippet 5-1: Examples of Valid Constructors](#)

Formatted: Caption

447

6 Implementation Scopes and Lifecycle Callbacks

448 The Java implementation type supports all of the scopes defined in the SCA-J Common Annotations and
449 APIs Specification: STATELESS and COMPOSITE. The SCA runtime MUST support the STATELESS
450 and COMPOSITE implementation scopes. [JCI60001]

451 Implementations specify their scope through the use of the @Scope annotation as shown in Snippet 6-1:

452

```
453 @Scope("COMPOSITE")  
454 public class ClientComponentImpl implements Client {  
455     // ...  
456 }
```

Formatted: Caption

457 [Snippet 6-1: Specifying the Scope of an Implementation](#)

458

459 When the @Scope annotation is not specified on an implementation class, its scope is defaulted to
460 STATELESS.

461 A Java component implementation specifies init and destroy methods by using the @Init and @Destroy
462 annotations respectively, as described in the SCA-J Common Annotations and APIs specification
463 [JAVACAA].

464 For example:

```
465 public class ClientComponentImpl implements Client {  
466  
467     @Init  
468     public void init() {  
469         //...  
470     }  
471  
472     @Destroy  
473     public void destroy() {  
474         //...  
475     }  
476 }
```

Formatted: Caption

477 [Snippet 6-2: Example Init and Destroy Methods](#)

478 **7 Accessing a Callback Service**

479 Java implementation classes that implement a service which has an associated callback interface can
480 use the `@Callback` annotation to have a reference to the callback service associated with the current
481 invocation injected on a field or injected via a setter method.

482 As an alternative to callback injection, references to the callback service can be accessed dynamically
483 through the API methods `RequestContext.getCallback()` and `RequestContext.getCallbackReference()` as
484 described in the SCA-J Common Annotations and APIs Specification [JAVACAA].

485

8 Component Type of a Java Implementation

486 An SCA runtime MUST introspect the componentType of a Java implementation class following the rules
487 defined in the section "Component Type of a Java Implementation". [JCI80001]

488 The component type of a Java Implementation is introspected from the implementation class using the
489 rules:

Deleted: as follows

490 A <service/> element exists for each interface or implementation class identified by a @Service
491 annotation:

Deleted: ¶

- 492 • name attribute is the simple name of the interface or implementation class (i.e., without the package
493 name)
- 494 • requires attribute is omitted unless the service implementation class is annotated with general or
495 specific intent annotations - in this case, the requires attribute is present with a value equivalent to the
496 intents declared by the service implementation class.
- 497 • policySets attribute is omitted unless the service implementation class is annotated with @PolicySets
498 - in this case, the policySets attribute is present with a value equivalent to the policy sets declared by
499 the @PolicySets annotation.
- 500 • <interface.java> child element is present with the interface attribute set to the fully qualified name of
501 the interface or implementation class identified by the @Service annotation. See the SCA-J Common
502 Annotations and APIs specification [JAVACAA] for a definition of how policy annotations on Java
503 interfaces, Java classes, and methods of Java interfaces are handled.
- 504 • remotable attribute of <interface.java> child element is omitted unless the service is defined by a Java
505 interface with no @Remotable annotation and the service implementation class is annotated with
506 @Remotable, in which case the <interface.java> element has remotable="true".
- 507 • binding child element is omitted
- 508 • callback child element is omitted

509 A <reference/> element exists for each @Reference annotation:

Deleted: ¶

- 510 • name attribute has the value of the name parameter of the @Reference annotation, if present,
511 otherwise it is the name of the field or the JavaBeans property name [JAVABEANS] corresponding to
512 the setter method name, depending on what element of the class is annotated by the @Reference
513 (note: for a constructor parameter, the @Reference annotation needs to have a name parameter)
- 514 • autowire attribute is omitted
- 515 • wiredByImpl attribute is omitted
- 516 • target attribute is omitted
- 517 • the multiplicity attribute is set according to the rules in section "@Reference" of the SCA Common
518 Annotations and APIs Specification [JAVACAA]
- 519 • requires attribute is omitted unless the field, setter method or parameter is also annotated with
520 general or specific intent annotations - in this case, the requires attribute is present with a value
521 equivalent to the intents declared by the Java reference.
- 522 • policySets attribute is omitted unless the field, setter method or parameter is also annotated with
523 @PolicySets - in this case, the policySets attribute is present with a value equivalent to the policy sets
524 declared by the @PolicySets annotation.
- 525 • <interface.java> child element with the interface attribute set to the fully qualified name of the
526 interface class which types the field or setter method or constructor parameter. See the SCA-J
527 Common Annotations and APIs specification [JAVACAA] for a definition of how policy annotations on
528 Java interfaces and methods of Java interfaces are handled.

529 • remotable attribute of <interface.java> child element is omitted unless the interface class has no
530 @Remotable annotation and there is a @Remotable annotation on the field, setter method or
531 constructor parameter, in which case the <interface.java> element has remotable="true".

532 • binding child element is omitted

533 • callback child element is omitted

534 | A <property/> element exists for each @Property annotation:

Deleted: ¶

535 • name attribute has the value of the name parameter of the @Property annotation, if present,
536 otherwise it is the name of the field or the JavaBeans property name [JAVABEANS] corresponding to
537 the setter method name, depending on what element of the class is annotated by the @Property
538 (note: for a constructor parameter, the @Property annotation needs to have a name parameter)

539 • value attribute is omitted

540 • type attribute which is set to the XML type implied by the JAXB mapping of the Java type of the field
541 or the Java type defined by the parameter of the setter method. Where the type of the field or of the
542 setter method is an array, the element type of the array is used. Where the type of the field or of the
543 setter method is a java.util.Collection, the parameterized type of the Collection or its member type is
544 used. If the JAXB mapping is to a global element rather than a type (JAXB @XMLRootElement
545 annotation), the type attribute is omitted. Note that JAXB mapping is the default mapping, but that
546 other mappings are possible, where supported by the SCA runtime
547 (for example, SDO). How such alternative mappings are indicated is not described in this
548 specification.

549 • element attribute is omitted unless the JAXB mapping of the Java type of the field or the Java type
550 defined by the parameter of the setter method is to a global element (JAXB @XMLRootElement
551 annotation). In this case, the element attribute has the value of the name of the XSD global element
552 implied by the JAXB mapping.

553 • many attribute is set according to the rules in section "@Property" of the SCA Common Annotations
554 and APIs Specification [JAVACAA].

555 • mustSupply attribute is set to "true" unless the @Property annotation has required=false, in which
556 case it is set to "false"

557 | An <implementation.java/> element exists if the service implementation class is annotated with general or
558 specific intent annotations or with @PolicySets:

Deleted: ¶

559 • requires attribute is omitted unless the service implementation class is annotated with general or
560 specific intent annotations - in this case, the requires attribute is present with a value equivalent to the
561 intents declared by the service implementation class.

562 • policySets attribute is omitted unless the service implementation class is annotated with @PolicySets
563 - in this case, the policySets attribute is present with a value equivalent to the policy sets declared by
564 the @PolicySets annotation.

565 **8.1 Component Type of an Implementation with no @Service, 566 @Reference or @Property Annotations**

567 The section defines the rules for determining the services of a Java component implementation that
568 contains no @Service annotations, no @Reference annotations, and no @Property annotations. If the
569 implementation class contains any @Service, @Reference or @Property annotations, the rules in this
570 section do not apply.

571 | The SCA services offered by the implementation class are defined using the rules:

Deleted: as follows

572 • either: one service for each of the interfaces implemented by the class where the interface is
573 annotated with @Remotable.

574 • or: if the class implements zero interfaces where the interface is annotated with @Remotable, then
575 by default the implementation offers a single local service whose type is the implementation class
576 itself

- 577 A <service/> element exists for each service identified in this way:
- 578 • name attribute is the simple name of the interface or the simple name of the class
- 579 • requires attribute is omitted unless the service implementation class is annotated with general or
- 580 specific intent annotations - in this case, the requires attribute is present with a value equivalent to the
- 581 intents declared by the service implementation class.
- 582 • policySets attribute is omitted unless the service implementation class is annotated with @PolicySets
- 583 - in this case, the policySets attribute is present with a value equivalent to the policy sets declared by
- 584 the @PolicySets annotation.
- 585 • <interface.java> child element is present with the interface attribute set to the fully qualified name of
- 586 the interface class or to the fully qualified name of the class itself. See the SCA-J Common
- 587 Annotations and APIs specification [JAVACAA] for a definition of how policy annotations on Java
- 588 interfaces, Java classes, and methods of Java interfaces are handled.
- 589 • remotable attribute of <interface.java> child element is omitted
- 590 • binding child element is omitted
- 591 • callback child element is omitted

592 ✓ The SCA properties and references of the implementation class are defined using the rules:

Deleted: ¶

Deleted: as follows

593 The following setter methods and fields are taken into consideration:

- 594 1. Public setter methods that are not part of the implementation of an SCA service (either explicitly
- 595 marked with @Service or implicitly defined as described above)
- 596 2. Public or protected fields unless there is a public setter method for the same name

597 ✓ An unannotated field or setter method is a **reference** if:

Deleted: ¶

- 598 • its type is an interface annotated with @Remotable
- 599 • its type is an array where the element type of the array is an interface annotated with @Remotable
- 600 • its type is a java.util.Collection where the parameterized type of the Collection or its member type is
- 601 an interface annotated with @Remotable

602 The reference in the component type has:

- 603 • name attribute with the value of the name of the field or the JavaBeans property name [JAVABEANS]
- 604 corresponding to the setter method name
- 605 • multiplicity attribute is (1..1) for the case where the type is an interface
- 606 multiplicity attribute is (1..n) for the cases where the type is an array or is a java.util.Collection
- 607 • <interface.java> child element with the interface attribute set to the fully qualified name of the
- 608 interface class which types the field or setter method. See the SCA-J Common Annotations and APIs
- 609 specification [JAVACAA] for a definition of how policy annotations on Java interfaces and methods of
- 610 Java interfaces are handled.
- 611 • remotable attribute of <interface.java> child element is omitted
- 612 • requires attribute is omitted unless the field or setter method is also annotated with general or
- 613 specific intent annotations - in this case, the requires attribute is present with a value equivalent
- 614 to the intents declared by the Java reference.
- 615 • policySets attribute is omitted unless the field or setter method is also annotated with
- 616 @PolicySets - in this case, the policySets attribute is present with a value equivalent to the policy
- 617 sets declared by the @PolicySets annotation.
- 618 • all other attributes and child elements of the reference are omitted

619 ✓ An unannotated field or setter method is a **property** if it is not a reference using the immediately

620 preceeding rules.

Deleted: ¶

Deleted: following

Deleted: above

621 For each property of this type, the component type has a property element with:

- 622 • name attribute with the value of the name of the field or the JavaBeans property name [JAVABEANS]
- 623 corresponding to the setter method name

- 624 • type attribute and element attribute are set as described for a property declared via a @Property
- 625 annotation, following the JAXB mapping of the Java type of the field or setter method by default. Note
- 626 that other mappings are possible, where supported by the SCA runtime (for example, SDO). How
- 627 such alternative mappings are indicated is not described in this specification.
- 628 • value attribute omitted
- 629 • many attribute set to "false" unless the type of the field or of the setter method is an array or a
- 630 java.util.Collection, in which case it is set to "true".
- 631 • mustSupply attribute set to true

632 8.2 Impact of JAX-WS Annotations on ComponentType

633 As described in the Java Common Annotations and APIs specification [JAVACAA], there are a number of
634 JAX-WS [JAX-WS] annotations that can affect the introspection and interpretation of Java classes and
635 Java interfaces. This section describes the effect of the JAX-WS annotations on the introspected
636 componentType of a Java implementation class.

637 8.2.1 @WebService

638 An interface or implementation class annotated with @WebService is treated as if it had an @Service
639 annotation:

- 640 • The value of the name property of the @WebService annotation is used as the name of the
- 641 <service/> element
- 642 • If the endpointInterface property of the @WebService annotation has a non-default value, then the
- 643 interface attribute of the <interface.java/> child element of the <service/> element is set to the
- 644 interface identified by the endpointInterface property.
- 645 • The <interface.java/> child element of the <service/> has the remotable attribute set to "true".
- 646 • If the wsdlLocation property of the @WebService annotation has a non-default value, then the
- 647 <service/> element has an <interface.wsdl/> child element instead of an <interface.java/> child
- 648 element. The value of the @interface attribute of the <interface.wsdl/> element is constructed by
- 649 pointing to the portType, in the WSDL definition pointed to by @wsdlLocation, which resulted from the
- 650 JAX-WS mapping for the annotated class or interface.

651 8.2.2 @WebMethod

- 652 • The value of the name property of the @WebMethod annotation is used when testing interface
- 653 compatibility.
- 654 • If the value of the exclude property of the @WebMethod annotation is "true", then the method is
- 655 excluded from the SCA interface.

656 8.2.3 @WebParam

- 657 • The value of the mode property of the @WebParam is considered when testing interface
- 658 compatibility.
- 659 • If the value of the header property of the @WebParam is "true", then the "SOAP" intent is added to
- 660 the requires annotation of the <service/> element.

661 8.2.4 @WebResult

- 662 • If the value of the header property of the @WebResult is "true", then the "SOAP" intent is added to
- 663 the requires annotation of the <service/> element.

664 8.2.5 @SOAPBinding

- 665 • If an interface or class is annotated with @SOAPBinding, then the “SOAP” intent is added to the
666 requires annotation of the <service/> element. The same is true if any method of the interface or
667 class is annotated with @SOAPBinding

668 8.2.6 @WebFault

- 669 • The value of the name property of the @WebFault annotation is used when testing interface
670 compatibility.

671 8.2.7 @WebServiceProvider

672 An implementation class annotated with @WebServiceProvider is treated as if it had an @Service
673 annotation:

- 674 • The <interface.java/> child element of the <service/> has the remotable attribute set to "true".
675 • If the wsdlLocation property of the @WebServiceProvider annotation has a non-default value, then
676 the <service/> element has an <interface.wsdl/> child element instead of an <interface.java/> child
677 element. The value of the @interface attribute of the <interface.wsdl/> element is constructed by
678 pointing to the portType, in the WSDL definition pointed to by @wsdlLocation, which resulted from the
679 JAX-WS mapping for the annotated class or interface.

Deleted: ¶

680 8.2.8 Web Service Binding

681 By default, the JAX-WS specification requires that JAX-WS service implementation classes have
682 endpoints that are made available using the SOAP 1.1 HTTP WSDL binding which is denoted by the URL
683 <http://schemas.xmlsoap.org/wsdl/soap/http> [JAX-WS].

684 Therefore, the presence of **any** JAX-WS annotations in an SCA implementation or in an interface class
685 requires that any SCA services exposed by an implementation class are made available using the SOAP
686 1.1 HTTP WSDL binding by default. As a result, the respective <service/> elements in the component
687 type of the implementation class each have a <binding.ws/> subelement [WSBINDING] with its
688 @wsdlElement attribute set such that the SOAP 1.1 HTTP WSDL binding is used at runtime.

689 Note that JAX-WS annotations do not cause <reference/> elements in the component type of an
690 implementation class to have a <binding.ws/> subelement.

691 8.2.8.1 @BindingType

692 If the default WSDL binding is not acceptable for a <service/>, the JAX-WS @BindingType annotation
693 can be used to specify a different WSDL binding URL. If the JAX-WS @BindingType annotation is used,
694 then the <binding.ws/> subelement has its @wsdlElement attribute set such that the WSDL binding used
695 at runtime matches the value of the @BindingType annotation.

696 8.3 Component Type Introspection Examples

697 **Snippet 8-1** shows how intent annotations can be applied to service and reference interfaces and
698 methods as well as to a service implementation class.

Deleted: Example 8.1

```
699 // Service interface  
700 package test;  
701 import org.oasisopen.sca.annotation.Authentication;  
702 import org.oasisopen.sca.annotation.Confidentiality;  
703  
704 @Authentication  
705 public interface MyService {  
706     @Confidentiality  
707     void mymethod();  
708 }  
709
```

```

710 // Reference interface
711 package test;
712 import org.oasisopen.sca.annotation.Integrity;
713
714 public interface MyRefInt {
715     @Integrity
716     void mymethod1();
717 }
718
719 // Service implementation class
720 package test;
721 import static org.oasisopen.sca.Constants.SCA_PREFIX;
722 import org.oasisopen.sca.annotation.Confidentiality;
723 import org.oasisopen.sca.annotation.Reference;
724 import org.oasisopen.sca.annotation.Service;
725 @Service(MyService.class)
726 @Requires(SCA_PREFIX+"managedTransaction")
727 public class MyServiceImpl {
728     @Confidentiality
729     @Reference
730     protected MyRefInt myRef;
731
732     public void mymethod() {...}
733 }

```

Deleted: Example 8.1. Intent annotations on Java interfaces, methods, and implementations.¶

734 [Snippet 8-1: Intent Annotations on Java Interfaces, Methods, and Implementations.](#)

735
736 [Snippet 8-2](#) shows the introspected component type that is produced by applying the component type
737 introspection rules to the interfaces and implementation from [Snippet 8-1](#).

Deleted: Example 8.2

Deleted: example 8.1

```

738 <componentType xmlns:sca=
739     "http://docs.oasis-open.org/ns/opencsa/sca/200912">
740     <implementation.java class="test.MyServiceImpl"
741         requires="sca:managedTransaction"/>
742     <service name="MyService" requires="sca:managedTransaction">
743         <interface.java interface="test.MyService"/>
744     </service>
745     <reference name="myRef" requires="sca:confidentiality">
746         <interface.java interface="test.MyRefInt"/>
747     </reference>
748 </componentType>

```

Deleted: Example 8.2. Introspected component type with intents.¶

749 [Snippet 8-2: Introspected Component Type with Intents](#)

750 8.4 Java Implementation with Conflicting Setter Methods

751 If a Java implementation class, with or without `@Property` and `@Reference` annotations, has more than
752 one setter method with the same JavaBeans property name [JAVABEANS] corresponding to the setter
753 method name, then if more than one method is inferred to set the same SCA property or to set the same
754 SCA reference, the SCA runtime MUST raise an error and MUST NOT instantiate the implementation
755 class. [JCI80002]

756 [Snippet 8-3](#) shows examples of illegal Java implementation due to the presence of more than one setter
757 method resulting in either an SCA property or an SCA reference with the same name:

Deleted: The following are

```

758
759 /** Illegal since two setter methods with same JavaBeans property name
760  * are annotated with @Property annotation. */
761 public class IllegalImpl1 {
762     // Setter method with upper case initial letter 'S'
763     @Property
764     public void setSomeProperty(String someProperty) {...}
765 }

```

```

766 // Setter method with lower case initial letter 's'
767 @Property
768 public void setsomeProperty(String someProperty) {...}
769 }
770
771 /** Illegal since setter methods with same JavaBeans property name
772 * are annotated with @Reference annotation. */
773 public class IllegalImpl2 {
774 // Setter method with upper case initial letter 'S'
775 @Reference
776 public void setSomeReference(SomeService service) {...}
777
778 // Setter method with lower case initial letter 's'
779 @Reference
780 public void setsomeReference(SomeService service) {...}
781 }
782
783 /** Illegal since two setter methods with same JavaBeans property name
784 * are resulting in an SCA property. Implementation has no @Property
785 * or @Reference annotations. */
786 public class IllegalImpl3 {
787 // Setter method with upper case initial letter 'S'
788 public void setSomeOtherProperty(String someProperty) {...}
789
790 // Setter method with lower case initial letter 's'
791 public void setsomeOtherProperty(String someProperty) {...}
792 }
793
794 /** Illegal since two setter methods with same JavaBeans property name
795 * are resulting in an SCA reference. Implementation has no @Property
796 * or @Reference annotations. */
797 public class IllegalImpl4 {
798 // Setter method with upper case initial letter 'S'
799 public void setSomeOtherReference(SomeService service) {...}
800
801 // Setter method with lower case initial letter 's'
802 public void setsomeOtherReference(SomeService service) {...}
803 }

```

Formatted: Caption

804 [Snippet 8-3: Example Conflicting Setter Methods](#)

806 [Snippet 8-4](#) is an example of a legal Java implementation in spite of the implementation class having two
807 setter methods with same JavaBeans property name [JAVABEANS] corresponding to the setter method
808 name:

Deleted: The following

```

810 /** Two setter methods with same JavaBeans property name, but one is
811 * annotated with @Property and the other is annotated with @Reference
812 * annotation. */
813 public class WeirdButLegalImpl {
814 // Setter method with upper case initial letter 'F'
815 @Property
816 public void setFoo(String foo) {...}
817
818 // Setter method with lower case initial letter 'f'
819 @Reference
820 public void setfoo(SomeService service) {...}
821 }

```

Formatted: Caption

822 [Snippet 8-4: Example of Valid Combination of Setter Methods](#)

823
824
825
826
827

9 Specifying the Java Implementation Type in an Assembly

Snippet 9-1 shows the pseudo-schema that defines the implementation element schema used for the Java implementation type:

Deleted: The following

Deleted: .

828
829
830

```
<implementation.java class="xs:NCName"
  requires="list of xs:QName"?
  policySets="list of xs:QName"?/>
```

Formatted: Caption, Indent: First line: 0 pt

831
832

Snippet 9-1: Pseudo-Schema for implementation.java

833

The implementation.java element has the attributes:

Deleted: following

834
835
836

- **class** : **NCName (1..1)** – the fully qualified name of the Java class of the implementation
- **requires** : **QName (0..n)** – a list of policy intents. See the [Policy Framework specification \[POLICY\]](#) for a description of this attribute.

837
838

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

839
840
841

The <implementation.java> element MUST conform to the schema defined in sca-implementation-java.xsd. [JCI90001]

Deleted: ¶

842
843
844

The fully qualified name of the Java class referenced by the @class attribute of <implementation.java/> MUST resolve to a Java class, using the artifact resolution rules defined in Section 10.2, that can be used as a Java component implementation. [JCI90002]

845
846

The Java class referenced by the @class attribute of <implementation.java/> MUST conform to Java SE version 5.0. [JCI90003]

847 10 Java Packaging and Deployment Model

848 The SCA Assembly Model Specification [ASSEMBLY] describes the basic packaging model for SCA
849 contributions in the chapter on Packaging and Deployment. This specification defines extensions to the
850 basic model for SCA contributions that contain Java component implementations.

851 The model for the import and export of Java classes follows the model for import-package and export-
852 package defined by the OSGi Service Platform Core Specification [OSGi Core]. Similar to an OSGi
853 bundle, an SCA contribution that contains Java classes represents a class loader boundary at runtime.
854 That is, classes are loaded by a contribution specific class loader such that all contributions with visibility
855 to those classes are using the same Class Objects in the JVM.

856 10.1 Contribution Metadata Extensions

857 SCA contributions can be self contained such that all the code and metadata needed to execute the
858 components defined by the contribution is contained within the contribution. However, in larger projects,
859 there is often a need to share artifacts across contributions. This is accomplished through the use of the
860 import and export extension points as defined in the sca-contribution.xml document. An SCA contribution
861 that needs to use a Java class from another contribution can declare the dependency via an

862 <import.java/> extension element, contained within a <contribution/> element, as [shown in Snippet 10-1](#):

```
863 <import.java package="xs:string" location="xs:anyURI" ?/>
```

864 [Snippet 10-1: Pseudo-Schema for import.java](#)

865

866 The import.java element has the attributes:

- 867 • **package** : **string (1..1)** – The name of one or more Java package(s) to use from another
868 contribution. Where there is more than one package, the package names are separated by a comma
869 ",".

870 The package can have a **version number range** appended to it, separated from the package name
871 by a semicolon ";" followed by the text "version=" and the version number range, for example:

872 package="com.acme.package1;version=1.4.1"

873 package="com.acme.package2;version=[1.2,1.3]"

874 Version number range follows the format defined in the OSGi Core specification [OSGi Core]:

875 [1.2,1.3] - enclosing square brackets - inclusive range meaning any version in the range from the
876 lowest to the highest, including the lowest and the highest

877 (1.3.1,2.4.1) - enclosing round brackets - exclusive range meaning any version in the range from the
878 lowest to the highest but not including the lowest or the highest.

879 1.4.1 - no enclosing brackets - implies any version at or later than the specified version number is
880 acceptable - equivalent to [1.4.1, infinity)

881 If no version is specified for an imported package, then it is assumed to have a version range of
882 [0.0.0, infinity) - ie any version is acceptable.

- 883 • **location** : **anyURI (0..1)** – The URI of the SCA contribution which is used to resolve the java
884 packages for this import.

885 Each Java package that is imported into the contribution MUST be included in one and only one
886 import.java element. [JCI100001] Multiple packages can be imported, either through specifying multiple
887 packages in the @package attribute or through the presence of multiple import.java elements.

Deleted: defined below

Formatted: Caption

Deleted: following

888 The SCA runtime MUST ensure that the package used to satisfy an import matches the package name,
889 the version number or version number range and (if present) the location specified on the import.java
890 element [JCI100002]

891 An SCA contribution that wants to allow a Java package to be used by another contribution can declare
892 the exposure via an <export.java/> extension element as [shown in Snippet 10-2](#):

Deleted: defined below

```
893 <export.java package="xs:string"/>
```

Formatted: Caption

894 [Snippet 10-2: Pseudo-Schema for export.java](#)

895

896 The export.java element has the attributes:

Deleted: following

897 • **package : string (1..1)** – The name of one or more Java package(s) to expose for sharing by another
898 contribution. Where there is more than one package, the package names are separated by a comma
899 ",".

900 The package can have a **version number** appended to it, separated from the package name by a
901 semicolon ";" followed by the text "version=" and the version number:

```
902 package="com.acme.package1;version=1.4.1"
```

903 The package can have a **uses directive** appended to it, separated from the package name by a
904 semicolon ";" followed by the text "uses=" which is then followed by a list of package names
905 contained within single quotes "" (needed as the list contains commas).

906 The uses directive indicates that the SCA runtime MUST ensure that any SCA contribution that
907 imports this package from this exporting contribution also imports the same version as is used by this
908 exporting contribution of any of the packages contained in the uses directive. [JCI100003] Typically,
909 the packages in the uses directive are packages used in the interface to the package being exported
910 (eg as parameters or as classes/interfaces that are extended by the exported package). Example:

```
911 package="com.acme.package1;uses='com.acme.package2,com.acme.package3'"
```

912 If no version information is specified for an exported package, the version defaults to 0.0.0.

913 If no uses directive is specified for an exported package, there is no requirement placed on a contribution
914 which imports the package to use any particular version of any other packages.

915 Each Java package that is exported from the contribution MUST be included in one and only one
916 export.java element. [JCI100004] Multiple packages can be exported, either through specifying multiple
917 packages in the @package attribute or through the presence of multiple export.java elements.

918 For example, a contribution that wants to:

919 use classes from the *some.package* package from another contribution (any version)

920 use classes of the *some.other.package* package from another contribution, at exactly version 2.0.0

921 expose the *my.package* package from its own contribution, with version set to 1.0.0

922 would specify an sca-contribution.xml file [shown in Snippet 10-3](#):

Deleted: as

Deleted: follows

923

```
924 <?xml version="1.0" encoding="UTF-8"?>  
925 <contribution xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200912>  
926 ...  
927 <import.java package="some.package"/>  
928 <import.java package="some.other.package;version=[2.0.0]"/>  
929 <export.java package="my.package;version=1.0.0"/>  
930 </contribution>
```

Formatted: Caption

931 [Snippet 10-3: Example Imports and Exports](#)

932

933 A Java package that is specified on an export element MUST be contained within the contribution
934 containing the export element. [JCI100007]

935

936 10.2 Java Artifact Resolution

937 The SCA runtime MUST ensure that within a contribution, Java classes are resolved according to the
938 following steps in the order specified:

- 939 1. If the contribution contains a Java Language specific resolution mechanism such as a classpath
940 declaration in the archive's manifest, then that mechanism is used first to resolve classes. If the
941 class is not found, then continue searching at step 2.
- 942 2. If the package of the Java class is specified in an import declaration then:
 - 943 a) if @location is specified, the location searched for the class is the contribution declared by
944 the @location attribute.
 - 945 b) if @location is not specified, the locations which are searched for the class are the
946 contribution(s) in the Domain which have export declarations for that package. If there is
947 more than one contribution exporting the package, then the contribution chosen is SCA
948 Runtime dependent, but is always the same contribution for all imports of the package.
- 949 If the Java package is not found, continue to step 3.
- 950 3. The contribution itself is searched using the archive resolution rules defined by the Java
951 Language.

952 [JCI100008]

953 10.3 Class Loader Model

954 The SCA runtime MUST ensure that the Java classes used by a contribution are all loaded by a class
955 loader that is unique for each contribution in the Domain. [JCI100010] The SCA runtime MUST ensure
956 that Java classes that are imported into a contribution are loaded by the exporting contribution's class
957 loader [JCI100011], as described in the section "Contribution Metadata Extensions"

958 For example, suppose contribution A using class loader ACL, imports package some.package from
959 contribution B that is using class loader BCL then the expression:

```
960 ACL.loadClass(importedClassName) == BCL.loadClass(importedClassName)
```

961 *Snippet 10-4: Example Class Loader Use*

962 evaluates to true.

963 The SCA runtime MUST set the thread context class loader of a component implementation class to the
964 class loader of its containing contribution. [JCI100009]

965

Formatted: Caption

Deleted: ¶

966 11 Conformance

967 The XML schema pointed to by the RDDL document at the namespace URI, defined by this specification,
968 are considered to be authoritative and take precedence over the XML schema defined in the appendix of
969 this document.

970 There are three categories of artifacts that this specification defines conformance for: SCA Java
971 Component Implementation Composite Document, SCA Java Component Implementation Contribution
972 Document and SCA Runtime.

Deleted: ¶

973 11.1 SCA Java Component Implementation Composite Document

974 An SCA Java Component Implementation Composite Document is an SCA Composite Document, as
975 defined by the SCA Assembly Model Specification Section 13.1 [ASSEMBLY], that uses the
976 <implementation.java> element. Such an SCA Java Component Implementation Composite Document
977 MUST be a conformant SCA Composite Document, as defined by [ASSEMBLY], and MUST comply with
978 the requirements specified in Section 9 of this specification.

Deleted: ¶

979 11.2 SCA Java Component Implementation Contribution Document

980 An SCA Java Component Implementation Contribution Document is an SCA Contribution Document, as
981 defined by the SCA Assembly Model specification Section 13.1 [ASSEMBLY], that uses the contribution
982 metadata extensions defined in Section 10. Such an SCA Java Component Implementation
983 Contribution document MUST be a conformant SCA Contribution Document, as defined by [ASSEMBLY],
984 and MUST comply with the requirements specified in Section 10 of this specification.

985 11.3 SCA Runtime

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

Deleted: following

- 987 1. The implementation MUST meet all the conformance requirements defined by the SCA Assembly
988 Model Specification [ASSEMBLY].
- 989 2. The implementation MUST reject an SCA Java Composite Document that does not conform to the
990 sca-implementation-java.xsd schema.
- 991 3. The implementation MUST reject an SCA Java Contribution Document that does not conform to the
992 sca-contribution-java.xsd schema.
- 993 4. The implementation MUST meet all the conformance requirements, specified in 'Section 11
994 Conformance', from the SCA-J Common Annotations and APIs Specification [JAVACAA].
- 995 5. This specification permits an implementation class to use any and all the APIs and annotations
996 defined in the SCA-J Common Annotations and APIs Specification [JAVACAA], therefore the
997 implementation MUST comply with all the statements in Appendix B: Conformance Items of
998 [JAVACAA], notably all mandatory statements have to be implemented.
- 999 6. The implementation MUST comply with all statements related to an SCA Runtime, specified in
1000 'Appendix B: Conformance Items' of this specification, notably all mandatory statements have to
1001 be implemented.

Deleted: ¶

1002

Deleted: ¶

1003

A. XML Schemas

1004

A.1 sca-contribution-java.xsd

```
1005 <?xml version="1.0" encoding="UTF-8"?>
1006 <!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved.
1007 OASIS trademark, IPR and other policies apply. -->
1008 <schema xmlns="http://www.w3.org/2001/XMLSchema"
1009 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1010 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1011 elementFormDefault="qualified">
1012
1013 <include schemaLocation="sca-contribution-1.1-cd04.xsd"/>
1014
1015 <!-- Import.java -->
1016 <element name="import.java" type="sca:JavaImportType"
1017 substitutionGroup="sca:importBase" />
1018 <complexType name="JavaImportType">
1019 <complexContent>
1020 <extension base="sca:Import">
1021 <attribute name="package" type="string" use="required"/>
1022 <attribute name="location" type="anyURI" use="optional"/>
1023 </extension>
1024 </complexContent>
1025 </complexType>
1026
1027 <!-- Export.java -->
1028 <element name="export.java" type="sca:JavaExportType"
1029 substitutionGroup="sca:exportBase" />
1030 <complexType name="JavaExportType">
1031 <complexContent>
1032 <extension base="sca:Export">
1033 <attribute name="package" type="string" use="required"/>
1034 </extension>
1035 </complexContent>
1036 </complexType>
1037
1038 </schema>
```

1039

A.2 sca-implementation-java.xsd

```
1040 <?xml version="1.0" encoding="UTF-8"?>
1041 <!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved.
1042 OASIS trademark, IPR and other policies apply. -->
1043 <schema xmlns="http://www.w3.org/2001/XMLSchema"
1044 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1045 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1046 elementFormDefault="qualified">
1047
1048 <include schemaLocation="sca-core-1.1-cd04.xsd"/>
1049
1050 <!-- Java Implementation -->
1051 <element name="implementation.java" type="sca:JavaImplementation"
1052 substitutionGroup="sca:implementation"/>
1053 <complexType name="JavaImplementation">
1054 <complexContent>
```

```
1055     <extension base="sca:Implementation">
1056         <sequence>
1057             <any namespace="##other" processContents="lax"
1058                 minOccurs="0" maxOccurs="unbounded"/>
1059         </sequence>
1060         <attribute name="class" type="NCName" use="required"/>
1061     </extension>
1062 </complexContent>
1063 </complexType>
1064
1065 </schema>
```

1066

B. Conformance Items

1067 This section contains a list of conformance items for the SCA Java Component Implementation
1068 specification.

1069

Conformance ID	Description
[JCI20001]	The services provided by a Java-based implementation MUST have an interface defined in one of the following ways: <ul style="list-style-type: none"> A Java interface A Java class A Java interface generated from a Web Services Description Language [WSDL] (WSDL) portType.
[JCI20002]	Java implementation classes MUST implement all the operations defined by the service interface.
[JCI40001]	For an unannotated field or setter method that is introspected as a property and where the Java type of the field or setter method is a JAXB [JAXB] annotated class, the SCA runtime MUST convert a property value specified by an SCA component definition into an instance of the property's Java type as defined by the XML to Java mapping in the JAXB specification [JAXB] with XML schema validation enabled.
[JCI50001]	A Java implementation class MUST provide a public or protected constructor that can be used by the SCA runtime to create the implementation instance.
[JCI50002]	The @Constructor annotation MUST NOT appear on more than one constructor.
[JCI50004]	The constructor to use for the creation of an implementation instance MUST be selected by the SCA runtime using the sequence: <ol style="list-style-type: none"> A declared constructor annotated with a @Constructor annotation. A declared constructor, all of whose parameters are annotated with either @Property or @Reference. A no-argument constructor.
[JCI50005]	In the absence of an @Constructor annotation, there MUST NOT be more than one constructor that has a non-empty parameter list with all parameters annotated with either @Property or @Reference.
[JCI60001]	The SCA runtime MUST support the STATELESS and COMPOSITE implementation scopes.
[JCI80001]	An SCA runtime MUST introspect the componentType of a Java implementation class following the rules defined in the section "Component Type of a Java Implementation".
[JCI80002]	If a Java implementation class, with or without @Property and @Reference annotations, has more than one setter method with the same JavaBeans property name [JAVABEANS] corresponding to the setter method name, then if more than one method is inferred to set the same SCA property or to set the same SCA reference, the SCA runtime MUST raise an error and MUST NOT instantiate the implementation class.

Formatted: Bullets and Numbering

[JCI90001]	The <implementation.java> element MUST conform to the schema defined in sca-implementation-java.xsd.
[JCI90002]	The fully qualified name of the Java class referenced by the @class attribute of <implementation.java/> MUST resolve to a Java class, using the artifact resolution rules defined in Section 10.2, that can be used as a Java component implementation.
[JCI90003]	The Java class referenced by the @class attribute of <implementation.java/> MUST conform to Java SE version 5.0.
[JCI100001]	Each Java package that is imported into the contribution MUST be included in one and only one import.java element.
[JCI100002]	The SCA runtime MUST ensure that the package used to satisfy an import matches the package name, the version number or version number range and (if present) the location specified on the import.java element.
[JCI100003]	The uses directive indicates that the SCA runtime MUST ensure that any SCA contribution that imports this package from this exporting contribution also imports the same version as is used by this exporting contribution of any of the packages contained in the uses directive.
[JCI100004]	Each Java package that is exported from the contribution MUST be included in one and only one export.java element.
[JCI100007]	A Java package that is specified on an export element MUST be contained within the contribution containing the export element.
[JCI100008]	<p>The SCA runtime MUST ensure that within a contribution, Java classes are resolved according to the following steps in the order specified:</p> <ol style="list-style-type: none"> 1. If the contribution contains a Java Language specific resolution mechanism such as a classpath declaration in the archive's manifest, then that mechanism is used first to resolve classes. If the class is not found, then continue searching at step 2. 2. If the package of the Java class is specified in an import declaration then: <ol style="list-style-type: none"> a) if @location is specified, the location searched for the class is the contribution declared by the @location attribute. b) if @location is not specified, the locations which are searched for the class are the contribution(s) in the Domain which have export declarations for that package. If there is more than one contribution exporting the package, then the contribution chosen is SCA Runtime dependent, but is always the same contribution for all imports of the package. <p>If the Java package is not found, continue to step 3.</p> 3. The contribution itself is searched using the archive resolution rules defined by the Java Language.
[JCI100009]	The SCA runtime MUST set the thread context class loader of a component implementation class to the class loader of its containing contribution.
[JCI100010]	The SCA runtime MUST ensure that the Java classes used by a contribution are all loaded by a class loader that is unique for each contribution in the Domain.
[JCI100011]	The SCA runtime MUST ensure that Java classes that are imported into a contribution are loaded by the exporting contribution's class loader

1071

C. Acknowledgements

1072 The following individuals have participated in the creation of this specification and are gratefully
1073 acknowledged:

1074 **Participants:**

Participant Name	Affiliation
Bryan Aupperle	IBM
Ron Barack	SAP AG*
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 Combellack	Avaya, Inc.
Jean-Sebastien Delfino	IBM
Mike Edwards	IBM
Raymond Feng	IBM
Bo Ji	Primeton Technologies, Inc.
Uday Joshi	Oracle Corporation
Anish Karmarkar	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	IBM
Luciano Resende	IBM
Michael Rowley	Active Endpoints, Inc.
Vladimir Savchenko	SAP AG*
Pradeep Simha	TIBCO Software Inc.
Raghav Srinivasan	Oracle Corporation

Scott Vorthmann
Feng Wang
Robin Yang

TIBCO Software Inc.
Primeton Technologies, Inc.
Primeton Technologies, Inc.

1075

1076



Deleted: ~~<#>Non-Normative Text¶~~
Formatted: Bullets and Numbering

1077

D. Revision History

1078 [optional; should not be included in OASIS Standards]

1079

Revision	Date	Editor	Changes Made
1	2007-09-26	Anish Karmarkar	Applied the OASIS template + related changes to the Submission
wd02	2008-12-16	David Booz	* Applied resolution for issue 55, 32 * Editorial cleanup to make a working draft - [1] style changed to [ASSEMBLY] - updated namespace references
wd03	2009-02-26	David Booz	<ul style="list-style-type: none"> Accepted all changes from wd02 Applied 60, 87, 117, 126, 123
wd04	2009-03-20	Mike Edwards	Accepted all changes from wd03 Issue 105 - RFC 2119 Language added - covers most of the specification. Accepted all changes after RFC 2119 language added. Editorial fix to ensure the term "class loader" is used consistently
wd05	2009-03-24	David Booz	Applied resolution for issues: 119, 137
wd06	2009-03-27	David Booz	Accepted all previous changes and applied issues 145,146,147,151
wd07	2009-04-06	David Booz	Editorial cleanup, namespace changes, changed XML encoding to UTF-8 in examples, applied 144
wd08	2009-04-27	David Booz	Applied issue 98, 152
wd09	2009-04-29	David Booz	Editorial fixes throughout (capitalization, quotes, fonts, spec references, etc.)
wd10	2009-04-30	David Booz	Editorial fixes, indentation, etc.
cd01	2009-05-04	David Booz	Final editorial fixes for CD and PRD
cd01-rev1	2009-08-12	David Booz	Editorial fixes, applied issues: 143,153,176
cd02-rev2	2009-09-14	David Booz	Applied issues: 157,162
cd02-rev3	2010-01-18	David Booz	Upgraded namespace to latest 200912 Applied issues: 168, 171, 181, 184, 186, 192,193

1080

1081