



Service Component Architecture POJO Component Implementation Specification Version 1.1

Committee Specification Draft ~~0304~~ /
Public Review Draft ~~0304~~

~~8 November 2010~~

15 August 2011

Specification URIs:

This ~~V~~version:

~~<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-csprd04.pdf> (Authoritative)~~
~~(Authoritative)~~
<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-csprd04.html>
<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-csprd04.doc>

Previous ~~V~~version:

~~<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-csprd03.pdf> (Authoritative)~~
~~(Authoritative)~~
<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-csprd03.html>
<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-csprd03.doc>

Latest ~~V~~version:

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

Technical Committee:

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

Chair(s):

Chairs:

David Booz (~~booz@us.ibm.com~~, ~~IBM~~), IBM
Anish Karmarkar (~~Anish.Karmarkar@oracle.com~~, ~~Oracle Corporation~~), Oracle

Editor(s):

Editors:

David Booz (~~booz@us.ibm.com~~, ~~IBM~~), IBM
Mike Edwards (~~mike_edwards@uk.ibm.com~~, ~~IBM~~), IBM
Anish Karmarkar (~~Anish.Karmarkar@oracle.com~~, ~~Oracle Corporation~~), Oracle

Related work:

This specification replaces or supersedes:

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

http://www.oesa.org/download/attachments/35/SCA_JavaComponentImplementation_V100.pdf?version=1

This specification is related to:

- [Service Component Architecture Assembly Model Specification Version 1.1. Latest version.](http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec.html)
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec.html>
- [SCA Policy Framework Version 1.1. Latest version.](http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1.html)
<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1.html>
- [Service Component Architecture SCA-J Common Annotations and APIs Specification Version 1.1. Latest version.](http://docs.oasis-open.org/opencsa/sca-j/sca-javaca-1.1-spec.html)
<http://docs.oasis-open.org/opencsa/sca-j/sca-javaca-1.1-spec.html>

Declared XML Namespace(s):namespaces:

- <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”~~ [“Send A Comment”](http://www.oasis-open.org/committees/sca-j/) 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-JAVACI]JavaCI-v1.1—OASIS Committee Specification Draft 03,~~

Service Component Architecture SCA-J POJO Component Implementation Specification Version 1.1, November 2010. [15 August 2011. OASIS Committee Specification Draft 04 / Public Review Draft 04.](http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-csprd04.html)

<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-csprd04.html>

Notices

Copyright © OASIS~~© 2005, 2010.~~ [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~~ [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"](#)", "["SCA"](#)" and "["Service Component Architecture"](#)" [are trademarks](#)" is a trademark of ~~OASIS~~ [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
1.3	Non-Normative References.....	6
1.4	Testcases.....	7
2	Service.....	8
2.1	Use of @Service.....	8
2.2	Local and Remotable Services.....	10
2.3	Introspecting Services Offered by a Java Implementation.....	12
2.4	Non-Blocking Service Operations.....	12
2.5	Callback Services.....	12
3	References.....	13
3.1	Reference Injection.....	13
3.2	Dynamic Reference Access.....	13
4	Properties.....	14
4.1	Property Injection.....	14
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.....	23
8.2.5	@SOAPBinding.....	23
8.2.6	@WebServiceProvider.....	23
8.2.7	Web Service Binding.....	23
8.3	Component Type Introspection Examples.....	24
8.4	Java Implementation with Conflicting Setter Methods.....	25
9	Specifying the Java Implementation Type in an Assembly.....	27
10	Java Packaging and Deployment Model.....	28
10.1	Contribution Metadata Extensions.....	28
10.2	Java Artifact Resolution.....	30
10.3	Class Loader Model.....	30
11	Conformance.....	31
11.1	SCA Java Component Implementation Composite Document.....	31
11.2	SCA Java Component Implementation Contribution Document.....	31
11.3	SCA Runtime.....	31
Appendix A.	XML Schemas.....	32

A.1 sca-contribution-java.xsd.....	32
A.2 sca-implementation-java.xsd.....	32
Appendix B. Conformance Items.....	34
Appendix C. Acknowledgements.....	36
Appendix D. 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] OASIS Committee [Specification Draft 0608](#), *SCA Assembly Model Specification Version 1.1*, [January 2010](#)~~May 2011~~.
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-spec-v1.1-cd08.pdf>
- [POLICY] OASIS Committee [Specification Draft 0405](#), *SCA Policy Framework Specification Version 1.1*, [September 2010](#)~~July 2011~~.
<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-spec-v1.1-cd05.pdf>
- [JAVACAA] OASIS Committee [Specification Draft 0406](#), *Service Component Architecture SCA-J Common Annotations and APIs Specification Version 1.1*, [February 2010](#)~~August 2011~~.
<http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-spec-v1.1-cd06.pdf>
- [WSDL] WSDL Specification, WSDL 1.1: <http://www.w3.org/TR/wsd1>
- [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] OASIS Committee [Specification Draft 0405](#), *SCA Web Service Binding Specification Version 1.1*, [May 2010](#)~~July 2011~~.
<http://docs.oasis-open.org/opencsa/sca-bindings/sca-wsbinding-spec-v1.1-csd05.pdf>

1.3 Non-Normative References

- [POJOTESTS] OASIS Committee [Specification Draft 02](#), *SCA-J POJO Component Implementation v1.1 TestCases*, [August 2011](#)
<http://docs.oasis-open.org/opencsa/sca-j/sca-j-pojo-ci-testcases-v1.1-csd02.pdf>

45

46 **1.4 Testcases**

47 The SCA-J POJO Component Implementation v1.1 TestCases [POJOTESTS] defines the TestCases for
48 the SCA-J POJO Component Implementation specification. The TestCases represent a series of tests
49 that SCA runtimes are expected to pass in order to claim conformance to the requirements of the SCA-J
50 Component Implementation specification.

51 2 Service

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

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

- 55 • A Java interface
- 56 • A Java class
- 57 • A Java interface generated from a Web Services Description Language [WSDL] (WSDL) portType.

58 [JCI20001]

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

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

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

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

67 2.1 Use of @Service

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

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

75 Interface:

```
76 package services.hello;  
77  
78 public interface HelloService {  
79  
80     String hello(String message);  
81 }
```

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

83

84 Implementation class:

```
85 @Service(HelloService.class)  
86 public class HelloServiceImpl implements HelloService {  
87  
88     public String hello(String message) {  
89         ...  
90     }  
91 }
```

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

93

94 The XML representation of the component type for this implementation is shown in Snippet 2-3 for
95 illustrative purposes. There is no need to author the component type as it is introspected from the Java
96 class.

97

```
98 <?xml version="1.0" encoding="UTF-8"?>  
99 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
100  
101   <service name="HelloService">  
102     <interface.java interface="services.hello.HelloService"/>  
103   </service>  
104  
105 </componentType>
```

106 *Snippet 2-3: Effective Component Type for Implementation in Snippet 2-2*

107

108 Another possibility is to use the Java implementation class itself to define a service offered by a
109 component and the interface of the service. In this case, the `@Service` annotation can be used to
110 explicitly declare the implementation class defines the service offered by the implementation. In this case,
111 a component will only offer services declared by `@Service`. Snippet 2-4 illustrates this:

112

```
113 package services.hello;  
114  
115 @Service(HelloServiceImpl.class)  
116 public class HelloServiceImpl implements AnotherInterface {  
117  
118   public String hello(String message) {  
119     ...  
120   }  
121   ...  
122 }
```

123 *Snippet 2-4: Example of Java Class Defining a Service*

124

125 In Snippet 2-4, `HelloServiceImpl` offers one service as defined by the public methods of the
126 implementation class. The interface `AnotherInterface` in this case does not specify a service offered by
127 the component. Snippet 2-5 is an XML representation of the introspected component type:

```
128 <?xml version="1.0" encoding="UTF-8"?>  
129 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
130  
131   <service name="HelloServiceImpl">  
132     <interface.java interface="services.hello.HelloServiceImpl"/>  
133   </service>  
134  
135 </componentType>
```

136 *Snippet 2-5: Effective Component Type for Implementation in Snippet 2-4*

137

138 The `@Service` annotation can be used to specify multiple services offered by an implementation as in
139 Snippet 2-6:

140

```
141 @Service(interfaces={HelloService.class, AnotherInterface.class})  
142 public class HelloServiceImpl implements HelloService, AnotherInterface  
143 {  
144  
145   public String hello(String message) {  
146     ...  
147   }  
}
```

```
148     ...
149 }
```

150 *Snippet 2-6: Example of @Service Specifying Multiple Services*

151

152 Snippet 2-7 shows the introspected component type for this implementation.

```
153 <?xml version="1.0" encoding="UTF-8"?>
154 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
155
156     <service name="HelloService">
157         <interface.java interface="services.hello.HelloService"/>
158     </service>
159     <service name="AnotherService">
160         <interface.java interface="services.hello.AnotherService"/>
161     </service>
162
163 </componentType>
```

164 *Snippet 2-7: Effective Component Type for Implementation in Snippet 2-6*

165 2.2 Local and Remotable Services

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

170 Interface:

```
171 package services.hello;
172
173 @Remotable
174 public interface HelloService {
175
176     String hello(String message);
177 }
```

178 *Snippet 2-8: Example Remotable Interface*

179

180 Implementation class:

```
181 package services.hello;
182
183 @Service(HelloService.class)
184 public class HelloServiceImpl implements HelloService {
185
186     public String hello(String message) {
187         ...
188     }
189 }
```

190 *Snippet 2-9: Implementation for Remotable Interface*

191

192 Snippet 2-10 shows the introspected component type for this implementation.

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

199 *Snippet 2-10: Effective Component Type for Implementation in Snippet 2-9*

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

202 If a service is defined by a Java implementation class instead of a Java interface, the @Remotable
203 annotation can be used on the implementation class to indicate that the service is remotable. Snippet
204 2-11 demonstrates this:

```
205 package services.hello;  
206  
207 @Remotable  
208 @Service(HelloServiceImpl.class)  
209 public class HelloServiceImpl {  
210  
211     public String hello(String message) {  
212         ...  
213     }  
214 }
```

215 *Snippet 2-11: Remotable Interface Defined by a Class*

216

217 Snippet 2-12 shows the introspected component type for this implementation.

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

224 *Snippet 2-12: Effective Component Type for Implementation in Snippet 2-11*

225

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

228 It is also possible to use a Java interface with no @Remotable annotation to define an SCA service with
229 remotable semantics. In this case, the @Remotable annotation is placed on the service implementation
230 class, as shown in Snippet 2-13 and Snippet 2-14:

231 Interface:

```
232 package services.hello;  
233  
234 public interface HelloService {  
235  
236     String hello(String message);  
237 }
```

238 *Snippet 2-13: Interface without @Remotable*

239

240 Implementation class:

```
241 package services.hello;  
242  
243 @Remotable  
244 @Service(HelloService.class)  
245 public class HelloServiceImpl implements HelloService {  
246  
247     public String hello(String message) {  
248         ...  
249     }  
250 }
```

251 *Snippet 2-14: Interface Made Remotable with @Remotable on Implementation Class*

252

253 In this case the introspected component type for the implementation uses the @remotable attribute of the
254 <interface.java/> element, as shown in Snippet 2-15:

```
255 <?xml version="1.0" encoding="UTF-8"?>  
256 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
257   <service name="HelloService">  
258     <interface.java interface="services.hello.HelloService"  
259       remotable="true"/>  
260   </service>  
261 </componentType>
```

262 *Snippet 2-15: Effective Component Type for Implementation in Snippet 2-14*

263

264 An SCA service defined by a @Service annotation specifying a Java interface, with no @Remotable
265 annotation on either the interface or the service implementation class, is inferred to be a local service as
266 defined by the SCA Assembly Model Specification [ASSEMBLY]. Similarly, an SCA service defined by a
267 @Service annotation specifying a Java implementation class with no @Remotable annotation is inferred
268 to be a local service.

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

271 2.3 Introspecting Services Offered by a Java Implementation

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

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

281 2.4 Non-Blocking Service Operations

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

285 2.5 Callback Services

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

290 3 References

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

294 3.1 Reference Injection

295 A Java implementation type can explicitly specify its references through the use of the @Reference
296 annotation as in Snippet 3-1:

```
297  
298 public class ClientComponentImpl implements Client {  
299     private HelloService service;  
300  
301     @Reference  
302     public void setHelloService(HelloService service) {  
303         this.service = service;  
304     }  
305 }
```

306 *Snippet 3-1: Specifying a Reference*

307

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

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

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

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

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

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

330 3.2 Dynamic Reference Access

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

334 4 Properties

335 4.1 Property Injection

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

339 A Java implementation type can explicitly specify its properties through the use of the @Property
340 annotation as in Snippet 4-1:

```
341  
342 public class ClientComponentImpl implements Client {  
343     private int maxRetries;  
344  
345     @Property  
346     public void setMaxRetries(int maxRetries) {  
347         this.maxRetries = maxRetries;  
348     }  
349 }
```

350 *Snippet 4-1: Specifying a Property*

351

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

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

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

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

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

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 is a JAXB [JAXB] annotated class, the SCA runtime MUST convert a property
371 value specified by an SCA component definition into an instance of the property's Java type as defined by
372 the XML to Java mapping in the JAXB specification [JAXB] with XML schema validation enabled.
373 [JCI40001]

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

377 4.2 Dynamic Property Access

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

381 5 Implementation Instance Creation

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

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

- 390 1. A declared constructor annotated with a @Constructor annotation.
- 391 2. A declared constructor, all of whose parameters are annotated with either @Property or
392 @Reference.
- 393 3. A no-argument constructor.

394 [JCI50004]

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

396 In the absence of an @Constructor annotation, there MUST NOT be more than one constructor that has
397 a non-empty parameter list with all parameters annotated with either @Property or @Reference.
398 [JCI50005]

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

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

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

```
405 /** Constructor declared using @Constructor annotation */  
406 public class Impl1 {  
407     private String someProperty;  
408     @Constructor  
409     public Impl1( @Property("someProperty") String propval ) {...}  
410 }  
411  
412 /** Declared constructor unambiguously identifying all Property  
413  * and Reference values */  
414 public class Impl2 {  
415     private String someProperty;  
416     private SomeService someReference;  
417     public Impl2( @Property("someProperty") String a,  
418                 @Reference("someReference") SomeService b )  
419         {...}  
420 }  
421  
422 /** Declared constructor unambiguously identifying all Property  
423  * and Reference values plus an additional Property injected  
424  * via a setter method */  
425 public class Impl3 {  
426     private String someProperty;  
427     private String anotherProperty;  
428     private SomeService someReference;  
429     public Impl3( @Property("someProperty") String a,  
430                 @Reference("someReference") SomeService b)  
431         {...}  
432     @Property  
433     public void setAnotherProperty( String anotherProperty ) {...}
```

```
434     }
435
436     /** No-arg constructor */
437     public class Impl4 {
438         @Property
439         public String someProperty;
440         @Reference
441         public SomeService someReference;
442         public Impl4() {...}
443     }
444
445     /** Unannotated implementation with no-arg constructor */
446     public class Impl5 {
447         public String someProperty;
448         public SomeService someReference;
449         public Impl5() {...}
450     }
```

451 *Snippet 5-1: Examples of Valid Constructors*

452 6 Implementation Scopes and Lifecycle Callbacks

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

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

457

```
458 @Scope("COMPOSITE")  
459 public class ClientComponentImpl implements Client {  
460     // ...  
461 }
```

462 *Snippet 6-1: Specifying the Scope of an Implementation*

463

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

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

469 For example:

```
470 public class ClientComponentImpl implements Client {  
471  
472     @Init  
473     public void init() {  
474         //...  
475     }  
476  
477     @Destroy  
478     public void destroy() {  
479         //...  
480     }  
481 }
```

482 *Snippet 6-2: Example Init and Destroy Methods*

483 **7 Accessing a Callback Service**

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

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

490 8 Component Type of a Java Implementation

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

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

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

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

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

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

- 534 • remotable attribute of <interface.java> child element is omitted unless the interface class has no
- 535 @Remotable annotation and there is a @Remotable annotation on the field, setter method or
- 536 constructor parameter, in which case the <interface.java> element has remotable="true".
- 537 • binding child element is omitted
- 538 • callback child element is omitted
- 539 A <property/> element exists for each @Property annotation:
 - 540 • name attribute has the value of the name parameter of the @Property annotation, if present,
 - 541 otherwise it is the name of the field or the JavaBeans property name [JAVABEANS] corresponding to
 - 542 the setter method name, depending on what element of the class is annotated by the @Property
 - 543 (note: for a constructor parameter, the @Property annotation needs to have a name parameter)
 - 544 • value attribute is omitted
 - 545 • type attribute which is set to the XML type implied by the JAXB mapping of the Java type of the field
 - 546 or the Java type defined by the parameter of the setter method. Where the type of the field or of the
 - 547 setter method is an array, the element type of the array is used. Where the type of the field or of the
 - 548 setter method is a java.util.Collection, the parameterized type of the Collection or its member type is
 - 549 used. If the JAXB mapping is to a global element rather than a type (JAXB @XMLRootElement
 - 550 annotation), the type attribute is omitted. Note that JAXB mapping is the default mapping, but that
 - 551 other mappings are possible, where supported by the SCA runtime
 - 552 (for example, SDO). How such alternative mappings are indicated is not described in this
 - 553 specification.
 - 554 • element attribute is omitted unless the JAXB mapping of the Java type of the field or the Java type
 - 555 defined by the parameter of the setter method is to a global element (JAXB @XMLRootElement
 - 556 annotation). In this case, the element attribute has the value of the name of the XSD global element
 - 557 implied by the JAXB mapping.
 - 558 • many attribute is set according to the rules in section “@Property” of the SCA Common Annotations
 - 559 and APIs Specification [JAVACAA].
 - 560 • mustSupply attribute is set to "true" unless the @Property annotation has required=false, in which
 - 561 case it is set to "false"
- 562 An <implementation.java/> element exists if the service implementation class is annotated with general or
- 563 specific intent annotations or with @PolicySets:
 - 564 • requires attribute is omitted unless the service implementation class is annotated with general or
 - 565 specific intent annotations - in this case, the requires attribute is present with a value equivalent to the
 - 566 intents declared by the service implementation class.
 - 567 • policySets attribute is omitted unless the service implementation class is annotated with @PolicySets
 - 568 - in this case, the policySets attribute is present with a value equivalent to the policy sets declared by
 - 569 the @PolicySets annotation.

570 **8.1 Component Type of an Implementation with no @Service,**

571 **@Reference or @Property Annotations**

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

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

- 577 • either: one service for each of the interfaces implemented by the class where the interface is
- 578 annotated with @Remotable.
- 579 • or: if the class implements zero interfaces where the interface is annotated with @Remotable, then
- 580 by default the implementation offers a single local service whose type is the implementation class
- 581 itself

582 A <service/> element exists for each service identified in this way:

583 • name attribute is the simple name of the interface or the simple name of the class

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

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

590 • <interface.java> child element is present with the interface attribute set to the fully qualified name of
591 the interface class or to the fully qualified name of the class itself. See the SCA-J Common
592 Annotations and APIs specification [JAVACAA] for a definition of how policy annotations on Java
593 interfaces, Java classes, and methods of Java interfaces are handled.

594 • remotable attribute of <interface.java> child element is omitted

595 • binding child element is omitted

596 • callback child element is omitted

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

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

599 1. Public setter methods that are not part of the implementation of an SCA service (either explicitly
600 marked with @Service or implicitly defined as described above)

601 2. Public or protected fields unless there is a public setter method for the same name

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

603 • its type is an interface annotated with @Remotable

604 • its type is an array where the element type of the array is an interface annotated with @Remotable

605 • its type is a java.util.Collection where the parameterized type of the Collection or its member type is
606 an interface annotated with @Remotable

607 The reference in the component type has:

608 • name attribute with the value of the name of the field or the JavaBeans property name [JAVABEANS]
609 corresponding to the setter method name

610 • multiplicity attribute is (1..1) for the case where the type is an interface
611 multiplicity attribute is (1..n) for the cases where the type is an array or is a java.util.Collection

612 • <interface.java> child element with the interface attribute set to the fully qualified name of the
613 interface class which types the field or setter method. See the SCA-J Common Annotations and APIs
614 specification [JAVACAA] for a definition of how policy annotations on Java interfaces and methods of
615 Java interfaces are handled.

616 • remotable attribute of <interface.java> child element is omitted

617 • requires attribute is omitted unless the field or setter method is also annotated with general or
618 specific intent annotations - in this case, the requires attribute is present with a value equivalent
619 to the intents declared by the Java reference.

620 • policySets attribute is omitted unless the field or setter method is also annotated with
621 @PolicySets - in this case, the policySets attribute is present with a value equivalent to the policy
622 sets declared by the @PolicySets annotation.

623 • all other attributes and child elements of the reference are omitted

624 An unannotated field or setter method is a **property** if it is not a reference using the immediately
625 preceding rules.

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

627 • name attribute with the value of the name of the field or the JavaBeans property name [JAVABEANS]
628 corresponding to the setter method name

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

637 8.2 Impact of JAX-WS Annotations on ComponentType

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

642 8.2.1 @WebService

643 An interface or implementation class annotated with `@WebService` is treated as if it had an `@Service`
644 annotation:

- 645 • The value of the name property of the `@WebService` annotation is used as the name of the
646 `<service/>` element
- 647 • If the endpointInterface property of the `@WebService` annotation has a non-default value, then the
648 interface attribute of the `<interface.java/>` child element of the `<service/>` element is set to the
649 interface identified by the endpointInterface property.
- 650 • The `<interface.java/>` child element of the `<service/>` has the remotable attribute set to “true”.
- 651 • If the wsdlLocation property of the `@WebService` annotation has a non-default value, then the
652 `<service/>` element has an `<interface.wsdl/>` child element instead of an `<interface.java/>` child
653 element. The value of the `@interface` attribute of the `<interface.wsdl/>` element is constructed by
654 pointing to the portType, in the WSDL definition pointed to by `@wsdlLocation`, which resulted from the
655 JAX-WS mapping for the annotated class or interface.
- 656 • If both the endpointInterface and wsdlLocation properties of the `@WebService` annotation have
657 default values and there is no `@Service` annotation, then the interface attribute of the
658 `<interface.java/>` child element of the `<service/>` element is set to the fully qualified name of the
659 interface or implementation class.

660 As noted in the the SCA-J Common Annotations and APIs Specification [JAVACAA], a service name
661 explicitly provided in a `@Service` annotation overrides any name defined by a `@WebService` annotation.

662 8.2.2 @WebMethod

- 663 • The value of the name property of the `@WebMethod` annotation is used when testing interface
664 compatibility.
- 665 • If the value of the exclude property of the `@WebMethod` annotation is “true”, then the method is
666 excluded from the SCA interface.

667 8.2.3 @WebParam

- 668 • The value of the mode property of the `@WebParam` is considered when testing interface
669 compatibility.
- 670 • If the value of the header property of the `@WebParam` is “true”, then the “SOAP” intent is added to
671 the requires attribute of the `<service/>` element.

672 8.2.4 @WebResult

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

675 8.2.5 @SOAPBinding

- 676 • If an interface or class is annotated with @SOAPBinding, then the "SOAP" intent is added to the
677 requires attribute of the <service/> element. The same is true if any method of the interface or class
678 is annotated with @SOAPBinding

679 8.2.6 @WebServiceProvider

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

- 682 • Where the Java implementation class implements a Java interface that is annotated with
683 @Remotable:
 - 684 ○ The @name attribute of the <service/> element in the component type is the simple name of
685 the Java interface class where the Java implementation class implements the Java interface
686 marked with @Remotable.
 - 687 ○ The <service/> element has a <interface.java/> subelement with an @interface set to the fully
688 qualified name of the Java interface class.
- 689 • Where the Java implementation class does not implement a Java interface that is annotated with
690 @Remotable:
 - 691 ○ The @name attribute of the <service/> element in the component type is the simple name of
692 the Java implementation class.
 - 693 ○ The <service/> element has a <interface.java/> subelement with an @interface set to the fully
694 qualified name of the Java implementation class and the @remotable attribute is set to "true".
- 695 • If the wsdlLocation property of the @WebServiceProvider annotation has a non-default value, then
696 the <service/> element has an <interface.wsdl/> child element instead of an <interface.java/> child
697 element. The value of the @interface attribute of the <interface.wsdl/> element is constructed by
698 pointing to the portType, in the WSDL definition pointed to by @wsdlLocation, which resulted from the
699 JAX-WS mapping for the annotated class or interface.

700 8.2.7 Web Service Binding

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

704 Therefore, the presence of **any** JAX-WS annotations in an SCA implementation or in an interface class
705 requires that any SCA services exposed by an implementation class are made available using the SOAP
706 1.1 HTTP WSDL binding by default. As a result, the respective <service/> elements in the component
707 type of the implementation class each have a <binding.ws/> subelement [WSBINDING] with the
708 SOAP.v1_1 intent added to the requires attribute of the <binding.ws/> subelement.

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

711 8.2.7.1 @BindingType

712 If the default WSDL binding is not acceptable for a <service/>, the JAX-WS @BindingType annotation
713 can be used to specify a different WSDL binding URL. If the JAX-WS @BindingType annotation is used,
714 then the set of intents added to the requires attribute of the <binding.ws/> subelement is based on the
715 value of the @BindingType annotation. Table 8-1 shows the mapping of the common binding types to
716 intents. For any other URI not listed in the table, the mapped intents are undefined.

WSDL Binding Type	Intent(s)
http://schemas.xmlsoap.org/wsdl/soap/http	SOAP.v1_1
http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true	SOAP.v1_1
http://www.w3.org/2003/05/soap/bindings/HTTP/	SOAP.v1_2
http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true	SOAP.v1_2
http://www.w3.org/2010/soapjms/	SOAP, JMS

718 Table 8-1: Intents for WSDL Bindings

719

8.3 Component Type Introspection Examples

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

```

722 // Service interface
723 package test;
724 import org.oasisopen.sca.annotation.Authentication;
725 import org.oasisopen.sca.annotation.Confidentiality;
726
727 @Authentication
728 public interface MyService {
729     @Confidentiality
730     void mymethod();
731 }
732
733 // Reference interface
734 package test;
735 import org.oasisopen.sca.annotation.Integrity;
736
737 public interface MyRefInt {
738     @Integrity
739     void mymethod1();
740 }
741
742 // Service implementation class
743 package test;
744 import static org.oasisopen.sca.Constants.SCA_PREFIX;
745 import org.oasisopen.sca.annotation.Confidentiality;
746 import org.oasisopen.sca.annotation.Reference;
747 import org.oasisopen.sca.annotation.Service;
748 @Service(MyService.class)
749 @Requires(SCA_PREFIX+"managedTransaction")
750 public class MyServiceImpl {
751     @Confidentiality
752     @Reference
753     protected MyRefInt myRef;
754
755     public void mymethod() {...}
756 }

```

757 *Snippet 8-1: Intent Annotations on Java Interfaces, Methods, and Implementations.*

758

759 Snippet 8-2 shows the introspected component type that is produced by applying the component type
 760 introspection rules to the interfaces and implementation from Snippet 8-1.

```

761 <componentType xmlns:sca=
762     "http://docs.oasis-open.org/ns/opencsa/sca/200912">

```



```

763     <implementation.java class="test.MyServiceImpl"
764         requires="sca:managedTransaction"/>
765     <service name="MyService" requires="sca:managedTransaction">
766         <interface.java interface="test.MyService"/>
767     </service>
768     <reference name="myRef" requires="sca:confidentiality">
769         <interface.java interface="test.MyRefInt"/>
770     </reference>
771 </componentType>

```

772 *Snippet 8-2: Introspected Component Type with Intents*

773 8.4 Java Implementation with Conflicting Setter Methods

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

779 Snippet 8-3 shows examples of illegal Java implementation due to the presence of more than one setter
780 method resulting in either an SCA property or an SCA reference with the same name:

781

```

782 /** Illegal since two setter methods with same JavaBeans property name
783  * are annotated with @Property annotation. */
784 public class IllegalImpl1 {
785     // Setter method with upper case initial letter 'S'
786     @Property
787     public void setSomeProperty(String someProperty) {...}
788
789     // Setter method with lower case initial letter 's'
790     @Property
791     public void setsomeProperty(String someProperty) {...}
792 }
793
794 /** Illegal since setter methods with same JavaBeans property name
795  * are annotated with @Reference annotation. */
796 public class IllegalImpl2 {
797     // Setter method with upper case initial letter 'S'
798     @Reference
799     public void setSomeReference(SomeService service) {...}
800
801     // Setter method with lower case initial letter 's'
802     @Reference
803     public void setsomeReference(SomeService service) {...}
804 }
805
806 /** Illegal since two setter methods with same JavaBeans property name
807  * are resulting in an SCA property. Implementation has no @Property
808  * or @Reference annotations. */
809 public class IllegalImpl3 {
810     // Setter method with upper case initial letter 'S'
811     public void setSomeOtherProperty(String someProperty) {...}
812
813     // Setter method with lower case initial letter 's'
814     public void setsomeOtherProperty(String someProperty) {...}
815 }
816
817 /** Illegal since two setter methods with same JavaBeans property name
818  * are resulting in an SCA reference. Implementation has no @Property
819  * or @Reference annotations. */
820 public class IllegalImpl4 {
821     // Setter method with upper case initial letter 'S'

```

```
822     public void setSomeOtherReference(SomeService service) {...}
823
824     // Setter method with lower case initial letter 's'
825     public void setsomeOtherReference(SomeService service) {...}
826 }
```

827 *Snippet 8-3: Example Conflicting Setter Methods*

828

829 Snippet 8-4 is an example of a legal Java implementation in spite of the implementation class having two
830 setter methods with same JavaBeans property name [JAVABEANS] corresponding to the setter method
831 name:

832

```
833 /** Two setter methods with same JavaBeans property name, but one is
834  * annotated with @Property and the other is annotated with @Reference
835  * annotation. */
836 public class WeirdButLegalImpl {
837     // Setter method with upper case initial letter 'F'
838     @Property
839     public void setFoo(String foo) {...}
840
841     // Setter method with lower case initial letter 'f'
842     @Reference
843     public void setfoo(SomeService service) {...}
844 }
```

845 *Snippet 8-4: Example of Valid Combination of Settter Methods*

846 9 Specifying the Java Implementation Type in an 847 Assembly

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

850

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

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

855

856 The implementation.java element has the attributes:

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

862 The <implementation.java> element MUST conform to the schema defined in sca-implementation-
863 java.xsd. [\[JCI90001\]](#)

864

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

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

870 10 Java Packaging and Deployment Model

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

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

879 10.1 Contribution Metadata Extensions

880 SCA contributions can be self contained such that all the code and metadata needed to execute the
881 components defined by the contribution is contained within the contribution. However, in larger projects,
882 there is often a need to share artifacts across contributions. This is accomplished through the use of the
883 import and export extension points as defined in the sca-contribution.xml document. An SCA contribution
884 that needs to use a Java class from another contribution can declare the dependency via an
885 <import.java/> extension element, contained within a <contribution/> element, as shown in Snippet 10-1:

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

887 *Snippet 10-1: Pseudo-Schema for import.java*

888

889 The import.java element has the attributes:

- 890 • **package : string (1..1)** – The name of one or more Java package(s) to use from another
891 contribution. Where there is more than one package, the package names are separated by a comma
892 ",".

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

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

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

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

898 [1.2,1.3] - enclosing square brackets - inclusive range meaning any version in the range from the
899 lowest to the highest, including the lowest and the highest

900 (1.3.1,2.4.1) - enclosing round brackets - exclusive range meaning any version in the range from the
901 lowest to the highest but not including the lowest or the highest.

902 1.4.1 - no enclosing brackets - implies any version at or later than the specified version number is
903 acceptable - equivalent to [1.4.1, infinity)

904 If no version is specified for an imported package, then it is assumed to have a version range of
905 [0.0.0, infinity) - ie any version is acceptable.

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

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

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

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

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

917 *Snippet 10-2: Pseudo-Schema for export.java*

918

919 The export.java element has the attributes:

920 • **package : string (1..1)** – The name of one or more Java package(s) to expose for sharing by another
921 contribution. Where there is more than one package, the package names are separated by a comma
922 ",".

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

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

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

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

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

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

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

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

941 For example, a contribution that wants to:

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

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

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

945 would specify an sca-contribution.xml file shown in Snippet 10-3 :

946

```
947 <?xml version="1.0" encoding="UTF-8"?>  
948 <contribution xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200912>  
949 ...  
950 <import.java package="some.package"/>  
951 <import.java package="some.other.package;version=[2.0.0]"/>  
952 <export.java package="my.package;version=1.0.0"/>  
953 </contribution>
```

954 *Snippet 10-3: Example Imports and Exports*

955

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

958

959 10.2 Java Artifact Resolution

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

962 1. If the contribution contains a Java Language specific resolution mechanism such as a classpath
963 declaration in the archive's manifest, then that mechanism is used first to resolve classes. If the
964 class is not found, then continue searching at step 2.

965 2. If the package of the Java class is specified in an import declaration then:

966 a) if @location is specified, the location searched for the class is the contribution declared by
967 the @location attribute.

968 b) if @location is not specified, the locations which are searched for the class are the
969 contribution(s) in the Domain which have export declarations for that package. If there is
970 more than one contribution exporting the package, then the contribution chosen is SCA
971 Runtime dependent, but is always the same contribution for all imports of the package.

972 If the Java package is not found, continue to step 3.

973 3. The contribution itself is searched using the archive resolution rules defined by the Java
974 Language.

975 [JCI100008]

976 10.3 Class Loader Model

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

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

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

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

985 evaluates to true.

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

988 11 Conformance

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

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

995 11.1 SCA Java Component Implementation Composite Document

996 An SCA Java Component Implementation Composite Document is an SCA Composite Document, as
997 defined by the SCA Assembly Model Specification Section 13.1 [ASSEMBLY], that uses the
998 <implementation.java> element. Such an SCA Java Component Implementation Composite Document
999 MUST be a conformant SCA Composite Document, as defined by [ASSEMBLY], and MUST comply with
1000 the requirements specified in Section 9 of this specification.

1001 11.2 SCA Java Component Implementation Contribution Document

1002 An SCA Java Component Implementation Contribution Document is an SCA Contribution Document, as
1003 defined by the SCA Assembly Model specification Section 13.1 [ASSEMBLY], that uses the contribution
1004 metadata extensions defined in Section 10. Such an SCA Java Component Implementation
1005 Contribution document MUST be a conformant SCA Contribution Document, as defined by [ASSEMBLY],
1006 and MUST comply with the requirements specified in Section 10 of this specification.

1007 11.3 SCA Runtime

1008 An implementation that claims to conform to this specification MUST meet the conditions:

- 1009 1. The implementation MUST meet all the conformance requirements defined by the SCA Assembly
1010 Model Specification [ASSEMBLY].
- 1011 2. The implementation MUST reject an SCA Java Composite Document that does not conform to the
1012 sca-implementation-java.xsd schema.
- 1013 3. The implementation MUST reject an SCA Java Contribution Document that does not conform to the
1014 sca-contribution-java.xsd schema.
- 1015 4. The implementation MUST meet all the conformance requirements, specified in 'Section 11
1016 Conformance', from the SCA-J Common Annotations and APIs Specification [JAVACAA].
- 1017 5. This specification permits an implementation class to use any and all the APIs and annotations
1018 defined in the SCA-J Common Annotations and APIs Specification [JAVACAA], therefore the
1019 implementation MUST comply with all the statements in Appendix B: Conformance Items of
1020 [JAVACAA], notably all mandatory statements have to be implemented.
- 1021 6. The implementation MUST comply with all statements related to an SCA Runtime, specified in
1022 'Appendix B: Conformance Items' of this specification, notably all mandatory statements have to
1023 be implemented.

1024

Appendix A. XML Schemas

1025

A.1 sca-contribution-java.xsd

1026

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

1027

```
<!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved.
```

1028

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

1029

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

1030

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

1031

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

1032

```
  elementFormDefault="qualified">
```

1033

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

1034

```
  <!-- Import.java -->
```

1035

```
  <element name="import.java" type="sca:JavaImportType"
```

1036

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

1037

```
  <complexType name="JavaImportType">
```

1038

```
    <complexContent>
```

1039

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

1040

```
        <attribute name="package" type="string" use="required"/>
```

1041

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

1042

```
      </extension>
```

1043

```
    </complexContent>
```

1044

```
  </complexType>
```

1045

```
  <!-- Export.java -->
```

1046

```
  <element name="export.java" type="sca:JavaExportType"
```

1047

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

1048

```
  <complexType name="JavaExportType">
```

1049

```
    <complexContent>
```

1050

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

1051

```
        <attribute name="package" type="string" use="required"/>
```

1052

```
      </extension>
```

1053

```
    </complexContent>
```

1054

```
  </complexType>
```

1055

```
</schema>
```

1060

A.2 sca-implementation-java.xsd

1061

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

1062

```
<!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved.
```

1063

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

1064

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

1065

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

1066

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

1067

```
  elementFormDefault="qualified">
```

1068

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

1069

```
  <!-- Java Implementation -->
```

1070

```
  <element name="implementation.java" type="sca:JavaImplementation"
```

1071

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

1072

```
  <complexType name="JavaImplementation">
```

1073


```
1075     <complexContent>
1076         <extension base="sca:Implementation">
1077             <sequence>
1078                 <any namespace="##other" processContents="lax"
1079                     minOccurs="0" maxOccurs="unbounded"/>
1080             </sequence>
1081             <attribute name="class" type="NCName" use="required"/>
1082         </extension>
1083     </complexContent>
1084 </complexType>
1085
1086 </schema>
```

1087

Appendix B. Conformance Items

1088 This section contains a list of conformance items for the SCA Java Component Implementation
1089 specification.

1090

Conformance ID	Description
[JCI20001]	<p>The services provided by a Java-based implementation MUST have an interface defined in one of the following ways:</p> <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]	<p>Java implementation classes MUST implement all the operations defined by the service interface.</p>
[JCI40001]	<p>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.</p>
[JCI50001]	<p>A Java implementation class MUST provide a public or protected constructor that can be used by the SCA runtime to create the implementation instance.</p>
[JCI50002]	<p>The @Constructor annotation MUST NOT appear on more than one constructor.</p>
[JCI50004]	<p>The constructor to use for the creation of an implementation instance MUST be selected by the SCA runtime using the sequence:</p> <ol style="list-style-type: none"> 7. A declared constructor annotated with a @Constructor annotation. 8. A declared constructor, all of whose parameters are annotated with either @Property or @Reference. 9. A no-argument constructor.
[JCI50005]	<p>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.</p>
[JCI60001]	<p>The SCA runtime MUST support the STATELESS and COMPOSITE implementation scopes.</p>
[JCI80001]	<p>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".</p>
[JCI80002]	<p>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.</p>

[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

1092

Appendix C. Acknowledgements

1093 The following individuals have participated in the creation of this specification and are gratefully
1094 acknowledged:

1095 **Participants:**

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 Combellack	Avaya, Inc.
Jean-Sebastien Delfino	IBM
Derek Dougans	Individual
Mike Edwards	IBM
Ant Elder	IBM
Raymond Feng	IBM
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	IBM
Luciano Resende	IBM

Michael Rowley
Vladimir Savchenko
Pradeep Simha
Raghav Srinivasan
Scott Vorthmann
Feng Wang

Paul Yang

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

1096

1097

Appendix D. Revision History

1098

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
cd01-rev2	2009-09-14	David Booz	Applied issues: 157,162
cd01-rev3	2010-01-18	David Booz	Upgraded namespace to latest 200912 Applied issues: 168, 171, 181, 184, 186, 192,193
cd01-rev4	2010-01-20	Bryan Aupperle	Editorial updates to match OASIS document standards
CD02	2010-02-02	David Booz	Editorial updates to produce Committee Draft

			All changes accepted
CD02-rev1	2010-07-13	David Booz	Applied Issue 197
CSD02-rev2	2010-11-04	David Booz	Applied Issue 203, 204, 212, 213 and prep for CSD03
CSD03	2010-11-08	OASIS TC Admin	Clean version
WD031	2011-06-20	Mike Edwards	Issue 231 - Sections 1.3, 1.4
WD032	2011-08-14	Anish Karmarkar	Clean up and reference updates

1099