



# Service Component Architecture POJO Component Implementation Specification Version 1.1

## Committee Draft 02 – rev1

13 July 2010

Deleted: 2

Deleted: February

### Specification URIs:

#### This Version:

<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-cd02.html>  
<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-cd02.doc>  
<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-cd02.pdf> (Authoritative)

#### Previous 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)

#### 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)

### 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>

Deleted: 2

Deleted: February

**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/>.

Deleted: 2

Deleted: February

---

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

Deleted: 2

Deleted: February

# Table of Contents

1	Introduction .....	6
1.1	Terminology .....	6
1.2	Normative References .....	6
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	@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

Deleted: 22

Deleted: 2

Deleted: February

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

Deleted: 2

Deleted: February

---

# 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 Draft 05, “SCA Assembly Model Specification Version 1.1”, January 2010.  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd05.pdf>
- [POLICY] OASIS, Committee Draft 02, “SCA Policy Framework Specification Version 1.1”, February 2009.  
<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.pdf>
- [JAVACAA] OASIS, Committee Draft 04, “Service Component Architecture SCA-J Common Annotations and APIs Specification Version 1.1”, February 2010.  
<http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd04.pdf>
- [WSDL] WSDL Specification, WSDL 1.1: <http://www.w3.org/TR/wsd>
- [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 Draft 03, “SCA Web Service Binding Specification Version 1.1”, July 2009.

Deleted: 2

Deleted: February

Deleted: 2

Deleted: February

## 48 2 Service

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

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

- 52 • A Java interface
- 53 • A Java class
- 54 • A Java interface generated from a Web Services Description Language [WSDL] (WSDL) portType.

Formatted: Bullets and Numbering

55 [JCI20001]

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

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

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

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

Deleted: The services provided by a Java-based implementation MUST have an interface defined in one of the following ways:  
<#>A Java interface ¶  
<#>A Java class ¶  
A Java interface generated from a Web Services Description Language [WSDL] (WSDL) portType.

### 64 2.1 Use of @Service

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

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

Deleted: Snippet 2-1

Deleted: Snippet 2-2

72 Interface:

```
73 package services.hello;  
74  
75 public interface HelloService {  
76     String hello(String message);  
77 }  
78
```

79 Snippet 2-1: Example Java Service Interface

Deleted: 2

81 Implementation class:

```
82 @Service(HelloService.class)  
83 public class HelloServiceImpl implements HelloService {  
84  
85     public String hello(String message) {  
86         ...  
87     }  
88 }
```

89 Snippet 2-2: Example Java Component Implementation

Deleted: 2

Deleted: 2

Deleted: February



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

Deleted: Snippet 2-3

```
94  
95 <?xml version="1.0" encoding="UTF-8"?>  
96 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
97  
98   <service name="HelloService">  
99     <interface.java interface="services.hello.HelloService"/>  
100   </service>  
101  
102 </componentType>
```

Deleted: 2

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

Deleted: Snippet 2-2

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

Deleted: Snippet 2-4

```
109  
110 package services.hello;  
111  
112 @Service(HelloServiceImpl.class)  
113 public class HelloServiceImpl implements AnotherInterface {  
114  
115   public String hello(String message) {  
116     ...  
117   }  
118   ...  
119 }
```

Deleted: 2

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

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

Deleted: Snippet 2-4

Deleted: Snippet 2-5

```
125 <?xml version="1.0" encoding="UTF-8"?>  
126 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
127  
128   <service name="HelloServiceImpl">  
129     <interface.java interface="services.hello.HelloServiceImpl"/>  
130   </service>  
131  
132 </componentType>
```

Deleted: 2

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

Deleted: Snippet 2-4

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

Deleted: Snippet 2-6

```
137  
138 @Service(interfaces={HelloService.class, AnotherInterface.class})  
139 public class HelloServiceImpl implements HelloService, AnotherInterface  
140 {  
141  
142   public String hello(String message) {  
143     ...  
144   }  
145 }
```

Deleted: 2

Deleted: February

```
144 }
145 ...
146 }
```

Deleted: 2

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

148  
149 [Snippet 2-7](#) shows the introspected component type for this implementation.

Deleted: Snippet 2-7

```
150 <?xml version="1.0" encoding="UTF-8"?>
151 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
152
153   <service name="HelloService">
154     <interface.java interface="services.hello.HelloService"/>
155   </service>
156   <service name="AnotherService">
157     <interface.java interface="services.hello.AnotherService"/>
158   </service>
159
160 </componentType>
```

Deleted: 2

Deleted: Snippet 2-6

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

## 162 2.2 Local and Remotable Services

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

Deleted: Snippet 2-8

Deleted: Snippet 2-9

167 Interface:

```
168 package services.hello;
169
170 @Remotable
171 public interface HelloService {
172
173   String hello(String message);
174 }
```

Deleted: 2

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

176  
177 Implementation class:

```
178 package services.hello;
179
180 @Service(HelloService.class)
181 public class HelloServiceImpl implements HelloService {
182
183   public String hello(String message) {
184     ...
185   }
186 }
```

Deleted: 2

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

188  
189 [Snippet 2-10](#) shows the introspected component type for this implementation.

Deleted: Snippet 2-10

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

Deleted: 2

Deleted: February

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

Deleted: 2

Deleted: [Snippet 2-9](#)

197  
198 The interface specified in the @interface attribute of the <interface.java/> element is implicitly remotable  
199 because the Java interface contains @Remotable.

200 If a service is defined by a Java implementation class instead of a Java interface, the @Remotable  
201 annotation can be used on the implementation class to indicate that the service is remotable. [Snippet](#)  
202 [2-11](#), demonstrates this:

Deleted: [Snippet 2-11](#)

```
203 package services.hello;  
204  
205 @Remotable  
206 @Service(HelloServiceImpl.class)  
207 public class HelloServiceImpl {  
208     public String hello(String message) {  
209         ...  
210     }  
211 }  
212
```

Deleted: 2

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

214  
215 [Snippet 2-12](#), shows the introspected component type for this implementation.

Deleted: [Snippet 2-12](#)

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

Deleted: 2

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

Deleted: [Snippet 2-11](#)

223  
224 The interface specified in the @interface attribute of the <interface.java/> element is implicitly remotable  
225 because the Java implementation class contains @Remotable.

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

Deleted: [Snippet 2-13](#)

Deleted: [Snippet 2-14](#)

229 Interface:

```
230 package services.hello;  
231  
232 public interface HelloService {  
233     String hello(String message);  
234 }  
235
```

Deleted: 2

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

237  
238 Implementation class:

```
239 package services.hello;  
240  
241 @Remotable  
242 @Service(HelloService.class)  
243 public class HelloServiceImpl implements HelloService {  
244     public String hello(String message) {  
245         ...  
246     }  
247
```

Deleted: 2

Deleted: February

248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287

```
}  
}
```

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

Deleted: 2

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

Deleted: Snippet 2-15

```
<?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"  
      remotable="true"/>  
  </service>  
</componentType>
```

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

Deleted: 2

Deleted: Snippet 2-14

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

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

### 2.3 Introspecting Services Offered by a Java Implementation

The services offered by a Java implementation class are determined through introspection, as defined in the section "Component Type of a Java Implementation".

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

### 2.4 Non-Blocking Service Operations

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

### 2.5 Callback Services

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

Deleted: 2

Deleted: February

## 288 3 References

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

### 292 3.1 Reference Injection

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

Deleted: Snippet 3-1

```
295  
296 public class ClientComponentImpl implements Client {  
297     private HelloService service;  
298  
299     @Reference  
300     public void setHelloService(HelloService service) {  
301         this.service = service;  
302     }  
303 }
```

Deleted: 3

304 *Snippet 3-1: Specifying a Reference*

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

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

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

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

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

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

### 328 3.2 Dynamic Reference Access

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

Deleted: 2

Deleted: February

## 332 4 Properties

### 333 4.1 Property Injection

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

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

Deleted: Snippet 4-1

```
339  
340 public class ClientComponentImpl implements Client {  
341     private int maxRetries;  
342  
343     @Property  
344     public void setMaxRetries(int maxRetries) {  
345         this.maxRetries = maxRetries;  
346     }  
347 }
```

Deleted: 4

348 *Snippet 4-1: Specifying a Property*

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

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

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

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

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

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

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

Deleted: 2

Deleted: February

375 **4.2 Dynamic Property Access**

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

Deleted: 2

Deleted: February

379

## 5 Implementation Instance Creation

380

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

Formatted: Pattern: Clear

Deleted: A Java implementation class MUST provide a public or protected constructor that can be used by the SCA runtime to create the implementation instance.

386

[The constructor to use for the creation of an implementation instance MUST be selected by the SCA runtime using the sequence:](#)

388

1. [A declared constructor annotated with a @Constructor annotation.](#)

389

2. [A declared constructor, all of whose parameters are annotated with either @Property or @Reference.](#)

390

3. [A no-argument constructor.](#)

391

Formatted: Bullets and Numbering

Deleted: The constructor to use for the creation of an implementation instance MUST be selected by the SCA runtime using the sequence:¶  
<#>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.

392

[JCI50004]

393

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

394

[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.](#)

395

[JCI50005]

396

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

399

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

402

[Snippet 5-1](#) shows examples of legal Java component constructor declarations:

403

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

Deleted: The @Constructor annotation MUST NOT appear on more than one constructor.

Deleted: 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.

Deleted: Snippet 5-1

430

Deleted: 2

Deleted: February



```
431     public void setAnotherProperty( String anotherProperty ) {...}
432 }
433
434 /** No-arg constructor */
435 public class Impl4 {
436     @Property
437     public String someProperty;
438     @Reference
439     public SomeService someReference;
440     public Impl4() {...}
441 }
442
443 /** Unannotated implementation with no-arg constructor */
444 public class Impl5 {
445     public String someProperty;
446     public SomeService someReference;
447     public Impl5() {...}
448 }
```

Deleted: 5

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

Deleted: 2

Deleted: February

450

## 6 Implementation Scopes and Lifecycle Callbacks

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

454 Implementations specify their scope through the use of the @Scope annotation as shown in [Snippet 6-1](#);

Deleted: Snippet 6-1

455

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

Deleted: 6

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

461

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

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

467 For example:

```
468 public class ClientComponentImpl implements Client {  
469  
470     @Init  
471     public void init() {  
472         //...  
473     }  
474  
475     @Destroy  
476     public void destroy() {  
477         //...  
478     }  
479 }
```

Deleted: 6

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

Deleted: 2

Deleted: February

---

481 **7 Accessing a Callback Service**

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

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

Deleted: 2

Deleted: February

488

## 8 Component Type of a Java Implementation

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

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

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

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

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

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

Deleted: 2

Deleted: February

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

## 568 **8.1 Component Type of an Implementation with no @Service,**

### 569 **@Reference or @Property Annotations**

570 The section defines the rules for determining the services of a Java component implementation that

571 contains no @Service annotations, no @Reference annotations, and no @Property annotations. If the

572 implementation class contains any @Service, @Reference or @Property annotations, the rules in this

573 section do not apply.

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

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

Deleted: 2

Deleted: February

- 580 A <service/> element exists for each service identified in this way:
- 581
- name attribute is the simple name of the interface or the simple name of the class
- 582
- requires attribute is omitted unless the service implementation class is annotated with general or
- 583 specific intent annotations - in this case, the requires attribute is present with a value equivalent to the
- 584 intents declared by the service implementation class.

585

  - policySets attribute is omitted unless the service implementation class is annotated with @PolicySets

586 - in this case, the policySets attribute is present with a value equivalent to the policy sets declared by

587 the @PolicySets annotation.

588

  - <interface.java> child element is present with the interface attribute set to the fully qualified name of

589 the interface class or to the fully qualified name of the class itself. See the SCA-J Common

590 Annotations and APIs specification [JAVACAA] for a definition of how policy annotations on Java

591 interfaces, Java classes, and methods of Java interfaces are handled.

592

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

593

  - binding child element is omitted

594

  - callback child element is omitted

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

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

- 597
1. Public setter methods that are not part of the implementation of an SCA service (either explicitly
- 598 marked with @Service or implicitly defined as described above)
- 599
2. Public or protected fields unless there is a public setter method for the same name

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

- 601
- its type is an interface annotated with @Remotable
- 602
- its type is an array where the element type of the array is an interface annotated with @Remotable
- 603
- its type is a java.util.Collection where the parameterized type of the Collection or its member type is
- 604 an interface annotated with @Remotable

605 The reference in the component type has:

- 606
- name attribute with the value of the name of the field or the JavaBeans property name [JAVABEANS]
- 607 corresponding to the setter method name
- 608
- multiplicity attribute is (1..1) for the case where the type is an interface
- 609 multiplicity attribute is (1..n) for the cases where the type is an array or is a java.util.Collection
- 610
- <interface.java> child element with the interface attribute set to the fully qualified name of the
- 611 interface class which types the field or setter method. See the SCA-J Common Annotations and APIs
- 612 specification [JAVACAA] for a definition of how policy annotations on Java interfaces and methods of
- 613 Java interfaces are handled.

614

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

615

  - requires attribute is omitted unless the field or setter method is also annotated with general or

616 specific intent annotations - in this case, the requires attribute is present with a value equivalent

617 to the intents declared by the Java reference.

618

  - policySets attribute is omitted unless the field or setter method is also annotated with

619 @PolicySets - in this case, the policySets attribute is present with a value equivalent to the policy

620 sets declared by the @PolicySets annotation.

621

  - all other attributes and child elements of the reference are omitted

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

623 preceding rules.

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

- 625
- name attribute with the value of the name of the field or the JavaBeans property name [JAVABEANS]
- 626 corresponding to the setter method name

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

## 635 8.2 Impact of JAX-WS Annotations on ComponentType

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

### 640 8.2.1 @WebService

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

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

Formatted: Font color: Auto

Formatted: Bullets and Numbering

### 658 8.2.2 @WebMethod

- 659 • The value of the name property of the @WebMethod annotation is used when testing interface  
660 compatibility.
- 661 • If the value of the exclude property of the @WebMethod annotation is "true", then the method is  
662 excluded from the SCA interface.

### 663 8.2.3 @WebParam

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

Deleted: 2

Deleted: February

## 668 8.2.4 @WebResult

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

## 671 8.2.5 @SOAPBinding

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

## 675 8.2.6 @WebFault

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

## 678 8.2.7 @WebServiceProvider

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

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

## 687 8.2.8 Web Service Binding

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

691 Therefore, the presence of **any** JAX-WS annotations in an SCA implementation or in an interface class  
692 requires that any SCA services exposed by an implementation class are made available using the SOAP  
693 1.1 HTTP WSDL binding by default. As a result, the respective <service/> elements in the component  
694 type of the implementation class each have a <binding.ws/> subelement [WSBINDING] with its  
695 @wsdlElement attribute set such that the SOAP 1.1 HTTP WSDL binding is used at runtime.

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

### 698 8.2.8.1 @BindingType

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

## 703 8.3 Component Type Introspection Examples

704 [Snippet 8-1](#) shows how intent annotations can be applied to service and reference interfaces and  
705 methods as well as to a service implementation class.

```
706 // Service interface  
707 package test;  
708 import org.oasisopen.sca.annotation.Authentication;  
709 import org.oasisopen.sca.annotation.Confidentiality;  
710
```

Deleted: Snippet 8-1

Deleted: 2

Deleted: February



```

711 @Authentication
712 public interface MyService {
713     @Confidentiality
714     void mymethod();
715 }
716
717 // Reference interface
718 package test;
719 import org.oasisopen.sca.annotation.Integrity;
720
721 public interface MyRefInt {
722     @Integrity
723     void mymethod1();
724 }
725
726 // Service implementation class
727 package test;
728 import static org.oasisopen.sca.Constants.SCA_PREFIX;
729 import org.oasisopen.sca.annotation.Confidentiality;
730 import org.oasisopen.sca.annotation.Reference;
731 import org.oasisopen.sca.annotation.Service;
732 @Service(MyService.class)
733 @Requires(SCA_PREFIX+"managedTransaction")
734 public class MyServiceImpl {
735     @Confidentiality
736     @Reference
737     protected MyRefInt myRef;
738
739     public void mymethod() {...}
740 }

```

Deleted: 8

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

742

743 [Snippet 8-2](#) shows the introspected component type that is produced by applying the component type

Deleted: Snippet 8-2

744 introspection rules to the interfaces and implementation from [Snippet 8-1](#).

Deleted: Snippet 8-1

```

745 <componentType xmlns:sca=
746     "http://docs.oasis-open.org/ns/opensca/sca/200912">
747     <implementation.java class="test.MyServiceImpl"
748         requires="sca:managedTransaction"/>
749     <service name="MyService" requires="sca:managedTransaction">
750         <interface.java interface="test.MyService"/>
751     </service>
752     <reference name="myRef" requires="sca:confidentiality">
753         <interface.java interface="test.MyRefInt"/>
754     </reference>
755 </componentType>

```

Formatted: English U.S.

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

Deleted: 8

## 757 8.4 Java Implementation with Conflicting Setter Methods

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

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

Deleted: Snippet 8-3

765

```

766 /** Illegal since two setter methods with same JavaBeans property name

```

Deleted: 2

Deleted: February

```

767 * are annotated with @Property annotation. */
768 public class IllegalImpl1 {
769     // Setter method with upper case initial letter 'S'
770     @Property
771     public void setSomeProperty(String someProperty) {...}
772
773     // Setter method with lower case initial letter 's'
774     @Property
775     public void setsomeProperty(String someProperty) {...}
776 }
777
778 /** Illegal since setter methods with same JavaBeans property name
779 * are annotated with @Reference annotation. */
780 public class IllegalImpl2 {
781     // Setter method with upper case initial letter 'S'
782     @Reference
783     public void setSomeReference(SomeService service) {...}
784
785     // Setter method with lower case initial letter 's'
786     @Reference
787     public void setsomeReference(SomeService service) {...}
788 }
789
790 /** Illegal since two setter methods with same JavaBeans property name
791 * are resulting in an SCA property. Implementation has no @Property
792 * or @Reference annotations. */
793 public class IllegalImpl3 {
794     // Setter method with upper case initial letter 'S'
795     public void setSomeOtherProperty(String someProperty) {...}
796
797     // Setter method with lower case initial letter 's'
798     public void setsomeOtherProperty(String someProperty) {...}
799 }
800
801 /** Illegal since two setter methods with same JavaBeans property name
802 * are resulting in an SCA reference. Implementation has no @Property
803 * or @Reference annotations. */
804 public class IllegalImpl4 {
805     // Setter method with upper case initial letter 'S'
806     public void setSomeOtherReference(SomeService service) {...}
807
808     // Setter method with lower case initial letter 's'
809     public void setsomeOtherReference(SomeService service) {...}
810 }

```

Deleted: 8

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

812

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

Deleted: Snippet 8-4

816

```

817 /** Two setter methods with same JavaBeans property name, but one is
818 * annotated with @Property and the other is annotated with @Reference
819 * annotation. */
820 public class WeirdButLegalImpl {
821     // Setter method with upper case initial letter 'F'
822     @Property
823     public void setFoo(String foo) {...}
824
825     // Setter method with lower case initial letter 'f'
826     @Reference
827     public void setfoo(SomeService service) {...}

```

Deleted: 2

Deleted: February

828

}

829

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

Deleted: 8

Deleted: 2

Deleted: February

830

## 9 Specifying the Java Implementation Type in an Assembly

831

832 | [Snippet 9-1](#), shows the pseudo-schema that defines the implementation element schema used for the  
833 Java implementation type:

Deleted: Snippet 9-1

834

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

Deleted: 9

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

839

840 The implementation.java element has the attributes:

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

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

848

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

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

Deleted: 2

Deleted: February

## 854 10 Java Packaging and Deployment Model

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

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

### 863 10.1 Contribution Metadata Extensions

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

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

Deleted: Snippet 10-1

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

Deleted: 10

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

872

873 The import.java element has the attributes:

- 874 • **package : string (1..1)** – The name of one or more Java package(s) to use from another  
875 contribution. Where there is more than one package, the package names are separated by a comma  
876 ", ".

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

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

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

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

882 [1.2,1.3] - enclosing square brackets - inclusive range meaning any version in the range from the  
883 lowest to the highest, including the lowest and the highest

884 (1.3.1,2.4.1) - enclosing round brackets - exclusive range meaning any version in the range from the  
885 lowest to the highest but not including the lowest or the highest.

886 1.4.1 - no enclosing brackets - implies any version at or later than the specified version number is  
887 acceptable - equivalent to [1.4.1, infinity)

888 If no version is specified for an imported package, then it is assumed to have a version range of  
889 [0.0.0, infinity) - ie any version is acceptable.

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

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

Deleted: 2

Deleted: February

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

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

Deleted: Snippet 10-2

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

Deleted: 10

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

902

903 The export.java element has the attributes:

904 • **package : string (1..1)** – The name of one or more Java package(s) to expose for sharing by another  
905 contribution. Where there is more than one package, the package names are separated by a comma  
906 ",."

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

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

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

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

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

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

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

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

925 For example, a contribution that wants to:

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

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

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

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

Deleted: Snippet 10-3

930

```
931 <?xml version="1.0" encoding="UTF-8"?>  
932 <contribution xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200912>  
933 ...  
934 <import.java package="some.package" />  
935 <import.java package="some.other.package;version=[2.0.0]" />  
936 <export.java package="my.package;version=1.0.0" />  
937 </contribution>
```

Deleted: 10

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

939

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

Deleted: 2

Deleted: February

## 943 10.2 Java Artifact Resolution

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

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

959 [JCI100008]

## 960 10.3 Class Loader Model

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

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

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

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

969 evaluates to true.

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

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

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:
  - 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.
- If the Java package is not found, continue to step 3.
3. The contribution itself is searched using the archive resolution rules defined by the Java Language.

Deleted: 10

Deleted: 2

Deleted: February

---

## 972 11 Conformance

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

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

### 979 11.1 SCA Java Component Implementation Composite Document

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

### 985 11.2 SCA Java Component Implementation Contribution Document

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

### 991 11.3 SCA Runtime

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

- 993 1. The implementation MUST meet all the conformance requirements defined by the SCA Assembly  
994 Model Specification [ASSEMBLY].
- 995 2. The implementation MUST reject an SCA Java Composite Document that does not conform to the  
996 sca-implementation-java.xsd schema.
- 997 3. The implementation MUST reject an SCA Java Contribution Document that does not conform to the  
998 sca-contribution-java.xsd schema.
- 999 4. The implementation MUST meet all the conformance requirements, specified in 'Section 11  
1000 Conformance', from the SCA-J Common Annotations and APIs Specification [JAVACAA].
- 1001 5. This specification permits an implementation class to use any and all the APIs and annotations  
1002 defined in the SCA-J Common Annotations and APIs Specification [JAVACAA], therefore the  
1003 implementation MUST comply with all the statements in Appendix B: Conformance Items of  
1004 [JAVACAA], notably all mandatory statements have to be implemented.
- 1005 6. The implementation MUST comply with all statements related to an SCA Runtime, specified in  
1006 'Appendix B: Conformance Items' of this specification, notably all mandatory statements have to  
1007 be implemented.

Deleted: 2

Deleted: February



1008

## A. XML Schemas

1009

### A.1 sca-contribution-java.xsd

1010

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved.
      OASIS trademark, IPR and other policies apply. -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  elementFormDefault="qualified">

  <include schemaLocation="sca-contribution-1.1-cd05.xsd"/>

  <!-- Import.java -->
  <element name="import.java" type="sca:JavaImportType"
    substitutionGroup="sca:importBase" />
  <complexType name="JavaImportType">
    <complexContent>
      <extension base="sca:Import">
        <attribute name="package" type="string" use="required"/>
        <attribute name="location" type="anyURI" use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <!-- Export.java -->
  <element name="export.java" type="sca:JavaExportType"
    substitutionGroup="sca:exportBase" />
  <complexType name="JavaExportType">
    <complexContent>
      <extension base="sca:Export">
        <attribute name="package" type="string" use="required"/>
      </extension>
    </complexContent>
  </complexType>

</schema>
```

1044

### A.2 sca-implementation-java.xsd

1045

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved.
      OASIS trademark, IPR and other policies apply. -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  elementFormDefault="qualified">

  <include schemaLocation="sca-core-1.1-cd05.xsd"/>

  <!-- Java Implementation -->
  <element name="implementation.java" type="sca:JavaImplementation"
    substitutionGroup="sca:implementation"/>
  <complexType name="JavaImplementation">
```

Deleted: 2

Deleted: February

```
1059     <complexContent>
1060         <extension base="sca:Implementation">
1061             <sequence>
1062                 <any namespace="##other" processContents="lax"
1063                     minOccurs="0" maxOccurs="unbounded" />
1064             </sequence>
1065             <attribute name="class" type="NCName" use="required" />
1066         </extension>
1067     </complexContent>
1068 </complexType>
1069
1070 </schema>
```

Deleted: 2

Deleted: February

1071  
1072  
1073  
1074

## B. Conformance Items

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

Conformance ID	Description
<a href="#">[JCI20001]</a>	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"> <li>A Java interface</li> <li>A Java class</li> <li>A Java interface generated from a Web Services Description Language [WSDL] (WSDL) portType.</li> </ul>
<a href="#">[JCI20002]</a>	Java implementation classes MUST implement all the operations defined by the service interface.
<a href="#">[JCI40001]</a>	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.
<a href="#">[JCI50001]</a>	A Java implementation class MUST provide a public or protected constructor that can be used by the SCA runtime to create the implementation instance.
<a href="#">[JCI50002]</a>	The @Constructor annotation MUST NOT appear on more than one constructor.
<a href="#">[JCI50004]</a>	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"> <li>A declared constructor annotated with a @Constructor annotation.</li> <li>A declared constructor, all of whose parameters are annotated with either @Property or @Reference.</li> <li>A no-argument constructor.</li> </ol>
<a href="#">[JCI50005]</a>	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.
<a href="#">[JCI60001]</a>	The SCA runtime MUST support the STATELESS and COMPOSITE implementation scopes.
<a href="#">[JCI80001]</a>	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".
<a href="#">[JCI80002]</a>	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: Complex Script Font: Arial, 10 pt  
Deleted: [JCI20001]

Formatted: Complex Script Font: Arial, 10 pt  
Deleted: [JCI20002]

Deleted: [JCI40001]  
Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt  
Deleted: [JCI50001]

Formatted: Complex Script Font: Arial, 10 pt

Deleted: [JCI50002]  
Formatted: Complex Script Font: Arial, 10 pt  
Deleted: [JCI50004]

Deleted: [JCI50005]  
Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt, Bold  
Deleted: [JCI60001]

Formatted: Complex Script Font: Arial, 10 pt  
Deleted: [JCI80001]

Formatted: Complex Script Font: Arial, 10 pt  
Deleted: [JCI80002]

Deleted: 2  
Deleted: February

<a href="#">[JCI90001]</a>	The <implementation.java> element MUST conform to the schema defined in sca-implementation-java.xsd.
<a href="#">[JCI90002]</a>	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.
<a href="#">[JCI90003]</a>	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.
<a href="#">[JCI100002]</a>	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.
<a href="#">[JCI100003]</a>	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.
<a href="#">[JCI100004]</a>	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.
<a href="#">[JCI100008]</a>	The SCA runtime MUST ensure that within a contribution, Java classes are resolved according to the following steps in the order specified: <ol style="list-style-type: none"> <li>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.</li> <li>2. If the package of the Java class is specified in an import declaration then: <ol style="list-style-type: none"> <li>a) if @location is specified, the location searched for the class is the contribution declared by the @location attribute.</li> <li>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.</li> </ol> <p>If the Java package is not found, continue to step 3.</p> </li> <li>3. The contribution itself is searched using the archive resolution rules defined by the Java Language.</li> </ol>
[JCI100009]	The SCA runtime MUST set the thread context class loader of a component implementation class to the class loader of its containing contribution.
<a href="#">[JCI100010]</a>	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.
<a href="#">[JCI100011]</a>	The SCA runtime MUST ensure that Java classes that are imported into a contribution are loaded by the exporting contribution's class loader

Formatted: Complex Script Font: Arial, 10 pt

Deleted: [JCI90001]

Formatted: Complex Script Font: Arial, 10 pt

Deleted: [JCI90002]

Deleted: [JCI90003]

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Deleted: [JCI100002]

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Deleted: [JCI100003]

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Deleted: [JCI100004]

Deleted: [JCI100008]

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Formatted: Complex Script Font: Arial, 10 pt

Deleted: [JCI100010]

Formatted: Complex Script Font: Arial, 10 pt

Deleted: [JCI100011]

Deleted: 2

Deleted: February

## 1076 C. Acknowledgements

1077 The following individuals have participated in the creation of this specification and are gratefully  
1078 acknowledged:

### 1079 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

Deleted: 2

Deleted: February

Scott Vorthmann  
Feng Wang  
Robin Yang

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

1080

Deleted: 2

Deleted: February

1081

## D. Revision History

1082 [optional; should not be included in OASIS Standards]

1083

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"> <li>Accepted all changes from wd02</li> <li>Applied 60, 87, 117, 126, 123</li> </ul>
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

Deleted: 2

Deleted: February

CD02	2010-02-02	David Booz	Editorial updates to produce Committee Draft All changes accepted
<a href="#">CD02-rev1</a>	<a href="#">2010-07-13</a>	<a href="#">David Booz</a>	<a href="#">Applied Issue 197</a>

1084

Deleted: 2  
Deleted: February