



# Service Component Architecture Java Component Implementation Specification Version 1.1

Working Draft **10**,

**30th** April 2009

Deleted: +simon

Inserted: +simon

Deleted: 8

Deleted: 27

## Specification URIs:

### This Version:

<http://docs.oasis-open.org/sca-j/sca-javaci-1.1-spec-wd08.html>  
<http://docs.oasis-open.org/sca-j/sca-javaci-1.1-spec-wd08.doc>  
<http://docs.oasis-open.org/sca-j/sca-javaci-1.1-spec-wd08.pdf>

### Previous Version:

### Latest Version:

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

### Latest Approved Version:

## Technical Committee:

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

## Chair(s):

David Booz, IBM  
Mark Combella, Avaya

## Editor(s):

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

## Related work:

This specification replaces or supersedes:

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

This specification is related to:

- Service Component Architecture Assembly Model Specification Version 1.1
- Service Component Architecture Policy Framework Specification Version 1.1
- [Service Component Architecture Java Common Annotations and APIs Specification Version 1.1](#)

Formatted: Bullets and Numbering

## Declared XML Namespace(s):

<http://docs.oasis-open.org/ns/opencsa/sca/200903>

Deleted: 08

Deleted: 27

**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 Java Common Annotations and APIs specification.

Deleted:

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: 08

Deleted: 27

---

## Notices

Copyright © OASIS® 2005, 2009. All Rights Reserved.

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

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

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

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

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

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

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

The names "OASIS", [insert specific trademarked names and abbreviations here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

Deleted: 08

Deleted: 27

# Table of Contents

1	Introduction .....	5
1.1	Terminology .....	5
1.2	Normative References .....	5
1.3	Non-Normative References .....	5
2	Service .....	6
2.1	Use of @Service .....	6
2.2	Local and Remotable services .....	8
2.3	Introspecting services offered by a Java implementation .....	8
2.4	Non-Blocking Service Operations .....	8
2.5	Callback Services .....	8
3	References .....	9
3.1	Reference Injection .....	9
3.2	Dynamic Reference Access .....	9
4	Properties .....	10
4.1	Property Injection .....	10
4.2	Dynamic Property Access .....	10
5	Implementation Instance Creation .....	11
6	Implementation Scopes and Lifecycle Callbacks .....	13
7	Accessing a Callback Service .....	14
8	Component Type of a Java Implementation .....	15
8.1	Component Type of an Implementation with no @Service annotations .....	16
8.2	ComponentType of an Implementation with no @Reference or @Property annotations .....	17
8.3	Component Type Introspection Examples .....	18
8.4	Java Implementation with conflicting setter methods .....	19
9	Specifying the Java Implementation Type in an Assembly .....	21
10	Java Packaging and Deployment Model .....	22
10.1	Contribution Metadata Extensions .....	22
10.2	Java Artifact Resolution .....	24
10.3	Class loader Model .....	24
11	Conformance .....	25
11.1	SCA Java Component Implementation Composite Document .....	25
11.2	SCA Java Component Implementation Contribution Document .....	25
11.3	SCA Runtime .....	25
A.	XML Schemas .....	26
A.1	sca-contribution-java.xsd .....	26
A.2	sca-implementation-java.xsd .....	26
B.	Conformance Items .....	28
C.	Acknowledgements .....	31
D.	Non-Normative Text .....	33
E.	Revision History .....	34

Deleted: 08

Deleted: 27

# 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 Java 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 Common Annotations and APIs specification are normative.

Deleted:

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

Deleted: in [JAVACAA]

## 1.1 Terminology

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

## 1.2 Normative References

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.

[ASSEMBLY] SCA Assembly Model Specification Version 1.1, <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec.pdf>

Formatted: English U.S.

Deleted: <http://www.oasis-open.org/committees/download.php/31722/sca-assembly-1.1-spec-cd03.pdf>

[POLICY] SCA Policy Framework Specification Version 1.1, <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.pdf>

Deleted: <http://www.oasis-open.org/committees/download.php/31608/sca-policy-1.1-spec-cd02.pdf>

[JAVACAA] SCA Java Common Annotations and APIs Specification Version 1.1, <http://docs.oasis-open.org/opencsa/sca-j-javacaa-1.1-spec-cd01.pdf>

Comment: Update to CD03/PRD link at the appropriate time.

[WSDL] WSDL Specification, WSDL 1.1: <http://www.w3.org/TR/wsdl>

[OSGi Core] OSGi Service Platform Core Specification, Version 4.0.1 <http://www.osgi.org/download/r4v41/r4.core.pdf>

[JAVABEANS] JavaBeans 1.01 Specification, <http://java.sun.com/javase/technologies/desktop/javabeans/api/>

## 1.3 Non-Normative References

TBD TBD

Deleted: 08

Deleted: 27

## 38 2 Service

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

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

- 42 • A Java interface
- 43 • A Java class
- 44 • A Java interface generated from a Web Services Description Language [WSDL] (WSDL)
- 45 portType.

46 [JCI20001]

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

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

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

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

### 56 2.1 Use of @Service

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

62 The following is an example of a Java service interface and a Java implementation, which provides a  
63 service using that interface:

64 Interface:

```
65 package services.hello;  
66  
67 public interface HelloService {  
68     String hello(String message);  
69 }  
70  
71
```

72 Implementation class:

```
73 @Service(HelloService.class)  
74 public class HelloServiceImpl implements HelloService {  
75  
76     public String hello(String message) {  
77         ...  
78     }  
79 }  
80
```

81 The XML representation of the component type for this implementation is shown below for illustrative  
82 purposes. There is no need to author the component type as it is introspected from the Java class.

Formatted: Indent: Before: 0 pt

Deleted:

Formatted: Bulleted + Level: 1 +  
Aligned at: 18 pt + Tab after: 0 pt  
+ Indent at: 36 pt

Formatted: English U.S.

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. 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. ¶

Formatted: Indent: Before: 0 pt

Formatted: Indent: Before: 0 pt

Deleted: WSDL 2 Java and Java 2 WSDL s

Formatted: Indent: Before: 0 pt

Deleted: ,

Formatted: Indent: Before: 18 pt

Formatted: French France

Formatted: Indent: Before: 0 pt

Deleted: ...

Formatted: Indent: Before: 36 pt,  
First line: 17.85 pt

Formatted: Indent: Before: 18 pt

Formatted: Indent: Before: 0 pt

Deleted: 08

Deleted: 27

```

83 |
84 | <?xml version="1.0" encoding="UTF-8"?>
85 | <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
86 |
87 |   <service name="HelloService">
88 |     <interface.java interface="services.hello.HelloService"/>
89 |   </service>
90 |
91 | </componentType>
92 |

```

Deleted: . . .

Formatted: Indent: Before: 36 pt, First line: 17.85 pt

Formatted: Indent: Before: 17.85 pt

Formatted: Indent: Before: 0 pt

Another possibility is to use the Java implementation class itself to define a service offered by a component and the interface of the service. In this case, the @Service annotation can be used to explicitly declare the implementation class defines the service offered by the implementation. In this case, a component will only offer services declared by @Service. The following illustrates this:

```

97 |
98 | package services.hello;
99 |
100 | @Service(HelloServiceImpl.class)
101 | public class HelloServiceImpl implements AnotherInterface {
102 |
103 |     public String hello(String message) {
104 |         ...
105 |     }
106 |     ...
107 | }
108 |

```

Formatted: Indent: Before: 0 pt

Deleted:

Formatted: Indent: Before: 36 pt, First line: 17.85 pt

Formatted: Indent: Before: 18 pt

Formatted: Indent: Before: 0 pt

Formatted: Indent: Before: 0 pt

In the above example, HelloServiceImpl offers one service as defined by the public methods of the implementation class. The interface AnotherInterface in this case does not specify a service offered by the component. The following is an XML representation of the introspected component type:

```

112 | <?xml version="1.0" encoding="UTF-8"?>
113 | <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
114 |
115 |   <service name="HelloServiceImpl">
116 |     <interface.java interface="services.hello.HelloServiceImpl"/>
117 |   </service>
118 |
119 | </componentType>
120 |

```

Formatted: Indent: Before: 0 pt

Deleted:

Formatted: Indent: Before: 36 pt, First line: 17.85 pt

Formatted: Indent: Before: 0 pt

Formatted: Indent: Before: 17.85 pt

Formatted: Indent: Before: 0 pt

The @Service annotation can be used to specify multiple services offered by an implementation as in the following example:

```

123 |
124 | @Service(interfaces={HelloService.class, AnotherInterface.class})
125 | public class HelloServiceImpl implements HelloService, AnotherInterface
126 | {
127 |
128 |     public String hello(String message) {
129 |         ...
130 |     }
131 |     ...
132 | }
133 |

```

Deleted:

Deleted:

Formatted: Indent: Before: 36 pt, First line: 17.85 pt

Deleted:

Formatted: Indent: First line: 36 pt

Formatted: Indent: Before: 0 pt

Formatted: Indent: Before: 18 pt

Formatted: Indent: Before: 0 pt

The following snippet shows the introspected component type for this implementation.

```

134 |
135 | <?xml version="1.0" encoding="UTF-8"?>

```

Deleted: 08

Deleted: 27

```

136 | <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
137 |
138 |   <service name="HelloService">
139 |     <interface.java interface="services.hello.HelloService"/>
140 |   </service>
141 |   <service name="AnotherService">
142 |     <interface.java interface="services.hello.AnotherService"/>
143 |   </service>
144 |
145 | </componentType>

```

- Formatted: Indent: First line: 17.85 pt
- Formatted: Indent: Before: 0 pt
- Deleted: .
- Formatted: Indent: Before: 36 pt, First line: 17.85 pt
- Formatted: Indent: Before: 0 pt
- Deleted: .
- Formatted: Indent: Before: 36 pt, First line: 17.85 pt
- Formatted: Indent: Before: 0 pt
- Formatted: Indent: Before: 17.85 pt
- Deleted: s
- Formatted: Indent: Before: 0 pt
- Deleted: SCA Assembly Specification

## 146 2.2 Local and Remotable Services

147 A Java service contract defined by an interface or implementation class uses the @Remotable  
 148 annotation to declare that the service follows the semantics of remotable services as defined by the  
 149 [SCA Assembly Model Specification \[ASSEMBLY\]](#). The following example demonstrates the use of the  
 150 @Remotable annotation:

```

151 | package services.hello;
152 |
153 | @Remotable
154 | public interface HelloService {
155 |
156 |     String hello(String message);
157 | }
158 |

```

- Formatted: Indent: Before: 17.85 pt
- Formatted: Indent: Before: 18 pt
- Formatted: Indent: Before: 0 pt

159 Unless annotated with a @Remotable annotation, a service defined by a Java interface or a Java  
 160 implementation class is inferred to be a local service as defined by the SCA Assembly Model  
 161 Specification [\[ASSEMBLY\]](#).

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

- Deleted:
- Deleted: s
- Deleted: o
- Deleted: i
- Formatted: Indent: Before: 0 pt, Don't adjust space between Latin and Asian text
- Formatted: Indent: Before: 0 pt

## 164 2.3 Introspecting Services Offered by a Java Implementation

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

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

## 173 2.4 Non-Blocking Service Operations

174 Service operations defined by a Java interface or by a Java implementation class can use the  
 175 @OneWay annotation to declare that the SCA runtime needs to honor non-blocking semantics as  
 176 defined by the SCA Assembly [Model Specification \[ASSEMBLY\]](#) when a client invokes the service  
 177 operation.

- Formatted: Indent: Before: 0 pt

## 178 2.5 Callback Services

179 A callback interface can be declared by using the @Callback annotation on the service interface or  
 180 Java implementation class as described in the Java Common Annotations and API [Specification \[JAVACAA\]](#). Alternatively, the @callbackInterface attribute of the <interface.java/> element can be  
 181 used to declare a callback interface.  
 182

- Formatted: Indent: Before: 0 pt
- Deleted: 08
- Deleted: 27



### 183 3 References

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

Formatted: Indent: Before: 0 pt

#### 187 3.1 Reference Injection

188 | A Java implementation type can explicitly specify its references through the use of the @Reference  
189 | annotation as in the following example:

Formatted: Indent: Before: 0 pt

```
190 |  
191 | public class ClientComponentImpl implements Client {  
192 |     private HelloService service;  
193 |  
194 |     @Reference  
195 |     public void setHelloService(HelloService service) {  
196 |         this.service = service;  
197 |     }  
198 | }  
199 |
```

Formatted: Indent: Before: 17.85 pt

Deleted: .

Deleted: .

Deleted: .

Deleted: .

Deleted: .

Formatted: Indent: Before: 36 pt, First line: 17.85 pt

Formatted: Indent: Before: 17.85 pt

Formatted: Indent: Before: 0 pt, First line: 17.85 pt

Formatted: Indent: Before: 0 pt

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

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

209 | If @Reference marks a parameter on a constructor, the SCA runtime provides the appropriate  
210 | implementation of the service reference contract as specified by the constructor parameter during  
211 | **creation** of an implementation instance of the Java class.

Deleted: instantiation

212 | **Except for constructor parameters,** references marked with the @Reference annotation can be  
213 | declared with required=false, as defined by the Java Common Annotations and APIs Specification  
214 | [JAVACAA] - i.e. the reference multiplicity is 0..1 or 0..n, where the implementation is designed to  
215 | cope with the reference not being wired to a target service.

Deleted: R

216 | In the case where a Java class contains no @Reference or @Property annotations, references are  
217 | determined by introspecting the implementation class as described in the section "**ComponentType of  
218 | an Implementation with no @Reference or @Property annotations**".

Formatted: Indent: Before: 0 pt, Don't adjust space between Latin and Asian text

#### 219 3.2 Dynamic Reference Access

220 | As an alternative to reference injection, service references can be accessed dynamically through the  
221 | API methods ComponentContext.getService() and ComponentContext.getServiceReference() methods  
222 | as described in the Java Common Annotations and APIs Specification [JAVACAA].

Formatted: Indent: Before: 0 pt

Deleted: 08

Deleted: 27

## 223 4 Properties

### 224 4.1 Property Injection

225 Properties can be obtained either through injection or through the ComponentContext API as defined  
226 in the SCA Java Common Annotations and APIs Specification [JAVACAA]. When possible, the preferred  
227 mechanism for accessing properties is through injection.

Formatted: Indent: Before: 0 pt

228 A Java implementation type can explicitly specify its properties through the use of the @Property  
229 annotation as in the following example:

```
230 public class ClientComponentImpl implements Client {  
231     private int maxRetries;  
232  
233     @Property  
234     public void setMaxRetries(int maxRetries) {  
235         this.maxRetries = maxRetries;  
236     }  
237 }  
238  
239
```

Formatted: Indent: Before: 17.85 pt

Deleted: .

Deleted: .

Deleted: .

Deleted: .

Deleted: .

Formatted: Indent: Before: 36 pt,  
First line: 17.85 pt

Formatted: Indent: Before: 17.85 pt

Formatted: Indent: Before: 17.85 pt,  
First line: 0 pt

Formatted: Indent: Before: 0 pt

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

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

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

Deleted: instantiation

250 Except for constructor parameters, properties marked with the @Property annotation can be declared  
251 with required=false as defined by the Java Common Annotations and APIs Specification [JAVACAA],  
252 i.e., the property mustSupply attribute is false and where the implementation is designed to cope with  
253 the component configuration not supplying a value for the property.

Deleted: P

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

Formatted: Indent: Before: 0 pt,  
Don't adjust space between Latin and  
Asian text

### 257 4.2 Dynamic Property Access

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

Formatted: Indent: Before: 0 pt

Deleted: 08

Deleted: 27

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  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309

## 5 Implementation Instance Creation

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.

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

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

[JCI50004]

The @Constructor annotation MUST only be specified on one constructor; the SCA container MUST raise an error if multiple constructors are annotated with @Constructor. [JCI50002]

The SCA runtime MUST raise an error if there are multiple constructors that are not annotated with @Constructor and have a non-empty parameter list with all parameters annotated with either @Property or @Reference. [JCI50005]

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.

Cyclic references between components MUST be handled by the SCA runtime in one of two ways:

- If any reference in the cycle is optional, then the container can inject a null value during construction, followed by injection of a reference to the target before invoking any service.
- The container can inject a proxy to the target service; invocation of methods on the proxy can result in a ServiceUnavailableException

[JCI50003]

The following are examples of legal Java component constructor declarations:

```
/** Constructor declared using @Constructor annotation */
public class Impl1 {
    private String someProperty;
    @Constructor
    public Impl1( @Property("someProperty") String propval ) {...}
}

/** Declared constructor unambiguously identifying all Property
 * and Reference values */
public class Impl2 {
    private String someProperty;
    private SomeService someReference;
    public Impl2( @Property("someProperty") String a,
                 @Reference("someReference") SomeService b )
    {...}
}

/** Declared constructor unambiguously identifying all Property
 * and Reference values plus an additional Property injected
 * via a setter method */
```

Formatted	...	[1]
Deleted: The constructor to us	...	[2]
Formatted	...	[3]
Formatted	...	[4]
Formatted	...	[5]
Deleted: ¶	...	[6]
Formatted	...	[7]
Inserted: .		
Deleted: .		
Formatted	...	[8]
Formatted	...	[9]
Formatted	...	[10]
Deleted: .		
Formatted	...	[11]
Deleted: .		
Formatted	...	[12]
Deleted: .		
Formatted	...	[13]
Deleted: .		
Formatted	...	[14]
Deleted: .		
Formatted	...	[15]
Formatted	...	[16]
Deleted: .		
Formatted	...	[17]
Deleted: */		
Inserted: */¶		
Formatted	...	[18]
Deleted: ./		
Formatted	...	[19]
Deleted: *		
Formatted	...	[20]
Deleted:		
Inserted: /**		
Formatted	...	[21]
Deleted: .		
Formatted	...	[22]
Deleted: .		
Formatted	...	[23]
Deleted: .		
Formatted	...	[24]
Deleted: .		
Formatted	...	[25]
Deleted: .		
Formatted	...	[26]
Deleted: .		
Formatted	...	[27]
Deleted: .		
Formatted	...	[28]
Formatted	...	[29]
Formatted	...	[30]
Formatted	...	[31]
Formatted	...	[32]
Formatted	...	[33]
Formatted	...	[34]

```

310 public class Impl3 {
311     private String someProperty;
312     private String anotherProperty;
313     private SomeService someReference;
314     public Impl3( @Property("someProperty") String a,
315                 @Reference("someReference") SomeService b)
316     {...}
317     @Property
318     public void setAnotherProperty( String anotherProperty ) {...}
319 }
320
321 /** No-arg constructor */
322 public class Impl4 {
323     @Property
324     public String someProperty;
325     @Reference
326     public SomeService someReference;
327     public Impl4() {...}
328 }
329
330 /** Unannotated implementation with no-arg constructor */
331 public class Impl5 {
332     public String someProperty;
333     public SomeService someReference;
334     public Impl5() {...}
335 }

```

Deleted: .
Formatted ... [39]
Deleted: .
Formatted ... [40]
Deleted: .
Formatted ... [41]
Deleted: .
Formatted ... [42]
Deleted: .
Formatted ... [43]
Deleted: .
Formatted ... [44]
Deleted: .
Formatted ... [45]
Deleted: .
Formatted ... [46]
Deleted: .
Formatted ... [47]
Deleted: .
Formatted ... [48]
Formatted ... [49]
Deleted: .
Formatted ... [50]
Deleted: .
Formatted ... [51]
Deleted: .
Formatted ... [52]
Deleted: .
Formatted ... [53]
Deleted: .
Formatted ... [54]
Deleted: .
Formatted ... [55]
Deleted: .
Formatted ... [56]
Deleted: .
Formatted ... [57]
Formatted ... [58]
Deleted: .
Formatted ... [59]
Deleted: .
Formatted ... [60]
Deleted: .
Formatted ... [61]
Deleted: .
Formatted ... [62]
Deleted: .
Formatted ... [63]
Deleted: .
Formatted ... [64]
Formatted ... [65]

336

## 6 Implementation Scopes and Lifecycle Callbacks

337 The Java implementation type supports all of the scopes defined in the Java Common Annotations and  
338 APIs Specification: STATELESS and COMPOSITE. **The SCA runtime MUST support the STATELESS and**  
339 **COMPOSITE implementation scopes.** [JC160001]

340 Implementations specify their scope through the use of the @Scope annotation as in:

```
341 @Scope("COMPOSITE")
342 public class ClientComponentImpl implements Client {
343     // ...
344 }
345
```

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

348 A Java component implementation specifies init and destroy **methods** by using the @Init and  
349 @Destroy annotations respectively, as described in the Java Common Annotations and APIs  
350 specification [JAVACAA].

351 For example:

```
352 public class ClientComponentImpl implements Client {
353     @Init
354     public void init() {
355         //...
356     }
357
358     @Destroy
359     public void destroy() {
360         //...
361     }
362 }
363
364
```

Formatted: Indent: Before: 0 pt

Formatted: Indent: Before: 17.85 pt

Deleted: .

Deleted: "

Deleted: "

Deleted: .

Formatted: Indent: Before: 17.85 pt, First line: 0 pt

Formatted: Indent: Before: 0 pt

Deleted: callbacks

Deleted: .

Formatted: Indent: Before: 17.85 pt

Formatted: Indent: Before: 17.85 pt, First line: 0 pt

Formatted: Indent: Before: 17.85 pt, First line: 18.15 pt

Deleted: .

Formatted: Indent: Before: 36 pt, First line: 17.85 pt

Formatted: Indent: Before: 17.85 pt, First line: 0 pt

Formatted: Indent: Before: 17.85 pt, First line: 18.15 pt

Deleted: .

Formatted: Indent: Before: 36 pt, First line: 17.85 pt

Formatted: Indent: Before: 17.85 pt, First line: 0 pt

Deleted: 08

Deleted: 27

365

## 7 Accessing a Callback Service

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

Formatted: Indent: Before: 0 pt

369 | As an alternative to callback injection, references to the callback service can be accessed dynamically  
370 through the API methods `RequestContext.getCallback()` and `RequestContext.getCallbackReference()`  
371 as described in the Java Common Annotations and APIs Specification [JAVACAA].

Deleted: 08

Deleted: 27

372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416

## 8 Component Type of a Java Implementation

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". [JC180001]

The component type of a Java Implementation is introspected from the implementation class as follows:

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

- name attribute is the simple name of the interface or implementation class (i.e., without the package name)
- requires attribute is omitted unless the service implementation class is annotated with general or specific intent annotations - in this case, the requires attribute is present with a value equivalent to the intents declared by the service implementation class.
- policySets attribute is omitted unless the service implementation class is annotated with @PolicySets - in this case, the policySets attribute is present with a value equivalent to the policy sets declared by the @PolicySets annotation.
- <interface.java> child element is present with the interface attribute set to the fully qualified name of the interface or implementation class identified by the @Service annotation. See the Java Common Annotations and APIs specification [JAVACAA] for a definition of how policy annotations on Java interfaces, Java classes, and methods of Java interfaces are handled.
- binding child element is omitted
- callback child element is omitted

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

- name attribute has the value of the name parameter of the @Reference annotation, if present, otherwise it is the name of the field or the JavaBeans property name [JAVABEANS] corresponding to the setter method name, depending on what element of the class is annotated by the @Reference (note: for a constructor parameter, the @Reference annotation needs to have a name parameter)
- autowire attribute is omitted
- wiredByImpl attribute is omitted
- target attribute is omitted
- a) where the type of the field, setter or constructor parameter is an interface, the multiplicity attribute is (1..1) unless the @Reference annotation contains required=false, in which case it is (0..1)
- b) where the type of the field, setter or parameter is an array or is a java.util.Collection, the multiplicity attribute is (1..n) unless the @Reference annotation contains required=false, in which case it is (0..n)
- requires attribute is omitted unless the field, setter method or parameter is also annotated with general or specific intent annotations - in this case, the requires attribute is present with a value equivalent to the intents declared by the Java reference.
- policySets attribute is omitted unless the field, setter method or parameter is also annotated with @PolicySets - in this case, the policySets attribute is present with a value equivalent to the policy sets declared by the @PolicySets annotation.
- <interface.java> child element with the interface attribute set to the fully qualified name of the interface class which types the field or setter method. See the Java Common Annotations and

Deleted: 08

Deleted: 27

417 APIs specification [JAVACAA] for a definition of how policy annotations on Java interfaces and  
418 methods of Java interfaces are handled.

- 419 • binding child element is omitted
- 420 • callback child element is omitted

421

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

- 423 • name attribute has the value of the name parameter of the @Property annotation, if present,  
424 otherwise it is the name of the field or the JavaBeans property name [JAVABEANS]  
425 corresponding to the setter method name, depending on what element of the class is annotated  
426 by the @Property (note: for a constructor parameter, the @Property annotation needs to have a  
427 name parameter)
- 428 • value attribute is omitted
- 429 • type attribute which is set to the XML type implied by the JAXB mapping of the Java type of the  
430 field or the Java type defined by the parameter of the setter method. Where the type of the field  
431 or of the setter method is an array, the element type of the array is used. Where the type of the  
432 field or of the setter method is a java.util.Collection, the parameterized type of the Collection or its  
433 member type is used. If the JAXB mapping is to a global element rather than a type (JAXB  
434 @XMLRootElement annotation), the type attribute is omitted.
- 435 • element attribute is omitted unless the JAXB mapping of the Java type of the field or the Java  
436 type defined by the parameter of the setter method is to a global element (JAXB  
437 @XMLRootElement annotation). In this case, the element attribute has the value of the name of  
438 the XSD global element implied by the JAXB mapping.
- 439 • many attribute is set to "false" unless the type of the field or of the setter method is an array or a  
440 java.util.Collection, in which case it is set to "true".
- 441 • mustSupply attribute is set to "true" unless the @Property annotation has required=false, in which  
442 case it is set to "false"

Deleted: n

443

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

- 446 • requires attribute is omitted unless the service implementation class is annotated with general or  
447 specific intent annotations - in this case, the requires attribute is present with a value equivalent  
448 to the intents declared by the service implementation class.
- 449 • policySets attribute is omitted unless the service implementation class is annotated with  
450 @PolicySets - in this case, the policySets attribute is present with a value equivalent to the policy  
451 sets declared by the @PolicySets annotation.

## 452 8.1 Component Type of an Implementation with no @Service 453 Annotations

Deleted: a

454 The section defines the rules for determining the services of a Java component implementation that does  
455 not explicitly declare them using the @Service annotation. Note that these rules apply only to  
456 implementation classes that contain **no** @Service annotations.

457 If there are no SCA services specified with the @Service annotation in an implementation class, the class  
458 offers:

- 459 • either: one Service for each of the interfaces implemented by the class where the interface is  
460 annotated with @Remotable.
- 461 • or: if the class implements zero interfaces where the interface is annotated with @Remotable,  
462 then by default the implementation offers a single local service whose type is the  
463 implementation class itself

Formatted: Indent: Before: 17.85 pt, Hanging: 17.85 pt, Tabs: Not at 54 pt

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

Deleted: 08

Deleted: 27



- 465 • name attribute is the simple name of the interface or the simple name of the class
- 466 • requires attribute is omitted unless the service implementation class is annotated with general or
- 467 specific intent annotations - in this case, the requires attribute is present with a value equivalent
- 468 to the intents declared by the service implementation class.
- 469 • policySets attribute is omitted unless the service implementation class is annotated with
- 470 @PolicySets - in this case, the policySets attribute is present with a value equivalent to the policy
- 471 sets declared by the @PolicySets annotation.
- 472 • <interface.java> child element is present with the interface attribute set to the fully qualified name
- 473 of the interface class or to the fully qualified name of the class itself. See the Java Common
- 474 Annotations and APIs specification [JAVACAA] for a definition of how policy annotations on Java
- 475 interfaces, Java classes, and methods of Java interfaces are handled.
- 476 • binding child element is omitted
- 477 • callback child element is omitted

## 478 8.2 ComponentType of an Implementation with no @Reference or

### 479 @Property Annotations

Deleted: a

480 The section defines the rules for determining the properties and the references of a Java component  
 481 implementation that does not explicitly declare them using the @Reference or the @Property  
 482 annotations. Note that these rules apply only to implementation classes that contain **no** @Reference  
 483 annotations **and no** @Property annotations.

484

485 In the absence of any @Property or @Reference annotations, the properties and references of an  
 486 implementation class are defined as follows:

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

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

Formatted: Indent: Before: 17.85 pt, Hanging: 17.85 pt

491

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

- 493 • its type is an interface annotated with @Remotable
- 494 • its type is an array where the element type of the array is an interface annotated with
- 495 @Remotable
- 496 • its type is a java.util.Collection where the parameterized type of the Collection or its member
- 497 type is an interface annotated with @Remotable

Formatted: Indent: Before: 17.85 pt, Hanging: 17.85 pt, Tabs: Not at 54 pt + 390 pt

498 The reference in the component type has:

- 499 • name attribute with the value of the name of the field or the JavaBeans property name  
 500 [JAVABEANS] corresponding to the setter method name
- 501 • multiplicity attribute is (1..1) for the case where the type is an interface  
 502 multiplicity attribute is (1..n) for the cases where the type is an array or is a  
 503 java.util.Collection
- 504 • <interface.java> child element with the interface attribute set to the fully qualified name of  
 505 the interface class which types the field or setter method. See the Java Common Annotations  
 506 and APIs specification [JAVACAA] for a definition of how policy annotations on Java interfaces  
 507 and methods of Java interfaces are handled.
- 508 • requires attribute is omitted unless the field or setter method is also annotated with general or  
 509 specific intent annotations - in this case, the requires attribute is present with a value  
 510 equivalent to the intents declared by the Java reference.

Formatted: Indent: Before: 17.85 pt, Hanging: 17.85 pt, Tabs: Not at 54 pt + 390 pt

Deleted: 08

Deleted: 27

- 511 | • policySets attribute is omitted unless the field or setter method is also annotated with
- 512 | • @PolicySets - in this case, the policySets attribute is present with a value equivalent to the
- 513 | • policy sets declared by the @PolicySets annotation.
- 514 | • all other attributes and child elements of the reference are omitted

515 |

516 | An unannotated field or setter method is a **property** if it is not a reference following the rules above.

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

- 518 | • name attribute with the value of the name of the field or the JavaBeans property name
- 519 | • [JAVABEANS] corresponding to the setter method name
- 520 | • type attribute and element attribute set as described for a property declared via a @Property
- 521 | • annotation
- 522 | • value attribute omitted
- 523 | • many attribute set to "false" unless the type of the field or of the setter method is an array or
- 524 | • a java.util.Collection, in which case it is set to "true".
- 525 | • mustSupply attribute set to true

Formatted: Indent: Before: 17.85 pt, Hanging: 17.85 pt, Tabs: Not at 54 pt + 390 pt

## 526 | 8.3 Component Type Introspection Examples

527 | Example 8.1 shows how intent annotations can be applied to service and reference interfaces and

528 | methods as well as to a service implementation class.

```

529 | // Service interface
530 | package test;
531 | import org.oasisopen.sca.annotation.Authentication;
532 | import org.oasisopen.sca.annotation.Confidentiality;
533 |
534 | @Authentication
535 | public interface MyService {
536 |     @Confidentiality
537 |     void mymethod();
538 | }
539 |
540 | // Reference interface
541 | package test;
542 | import org.oasisopen.sca.annotation.Integrity;
543 |
544 | public interface MyRefInt {
545 |     @Integrity
546 |     void mymethod1();
547 | }
548 |
549 | // Service implementation class
550 | package test;
551 | import static org.oasisopen.sca.Constants.SCA_PREFIX;
552 | import org.oasisopen.sca.annotation.Confidentiality;
553 | import org.oasisopen.sca.annotation.Reference;
554 | import org.oasisopen.sca.annotation.Service;
555 | @Service(MyService.class)
556 | @Requires(SCA_PREFIX+"managedTransaction")
557 | public class MyServiceImpl {
558 |     @Confidentiality
559 |     @Reference
560 |     protected MyRefInt myRef;
561 |
562 |     public void mymethod() { : : }

```

Deleted: 08

Deleted: 27

563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615

}  
Example 8.1. Intent annotations on Java interfaces, methods, and implementations.  
Example 8.2 shows the introspected component type that is produced by applying the component type introspection rules to the interfaces and implementation from example 8.1.

```
<componentType xmlns:sca=  
    "http://docs.oasis-open.org/ns/opencsa/sca/200903">  
    <implementation.java class="test.MyServiceImpl"  
        requires="sca:managedTransaction"/>  
    <service name="MyService" requires="sca:managedTransaction">  
        <interface.java interface="test.MyService"/>  
    </service>  
    <reference name="myRef" requires="sca:confidentiality">  
        <interface.java interface="test.MyRefInt"/>  
    </reference>  
</componentType>
```

Formatted: Indent: Before: 17.85 pt  
Deleted: c  
Deleted: s  
Deleted: m

Example 8.2. Introspected component type with intents.

### 8.4 Java Implementation with **C**onflicting **S**etter **M**ethods

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. [JC180002]

The following are examples of illegal Java implementation due to the presence of more than one setter method resulting in either an SCA property or an SCA reference with the same name:

```
/** Illegal since two setter methods with same JavaBeans property name  
 * are annotated with @Property annotation. */  
public class IllegalImpl1 {  
    // Setter method with upper case initial letter 'S'  
    @Property  
    public void setSomeProperty(String someProperty) {...}  
  
    // Setter method with lower case initial letter 's'  
    @Property  
    public void setsomeProperty(String someProperty) {...}  
}  
  
/** Illegal since setter methods with same JavaBeans property name  
 * are annotated with @Reference annotation. */  
public class IllegalImpl2 {  
    // Setter method with upper case initial letter 'S'  
    @Reference  
    public void setSomeReference(SomeService service) {...}  
  
    // Setter method with lower case initial letter 's'  
    @Reference  
    public void setsomeReference(SomeService service) {...}  
}  
  
/** Illegal since two setter methods with same JavaBeans property name  
 * are resulting in an SCA property. Implementation has no @Property  
 * or @Reference annotations. */  
public class IllegalImpl3 {
```

Formatted: Indent: Before: 17.85 pt  
Deleted:

Deleted: 08  
Deleted: 27

```

616     // Setter method with upper case initial letter 'S'
617     public void setSomeOtherProperty(String someProperty) {...}
618
619     // Setter method with lower case initial letter 's'
620     public void setsomeOtherProperty(String someProperty) {...}
621 }
622
623 /** Illegal since two setter methods with same JavaBeans property name
624 * are resulting in an SCA reference. Implementation has no @Property
625 * or @Reference annotations. */
626 public class IllegalImpl4 {
627     // Setter method with upper case initial letter 'S'
628     public void setSomeOtherReference(SomeService service) {...}
629
630     // Setter method with lower case initial letter 's'
631     public void setsomeOtherReference(SomeService service) {...}
632 }
633

```

634 The following is an example of a legal Java implementation in spite of the implementation class having  
635 two setter methods with same JavaBeans property name [JAVABEANS] corresponding to the setter  
636 method name:

```

637
638 /** Two setter methods with same JavaBeans property name, but one is
639 * annotated with @Property and the other is annotated with @Reference
640 * annotation. */
641 public class WeirdButLegalImpl {
642     // Setter method with upper case initial letter 'F'
643     @Property
644     public void setFoo(String foo) {...}
645
646     // Setter method with lower case initial letter 'f'
647     @Reference
648     public void setfoo(SomeService service) {...}
649 }
650

```

Formatted: Indent: Before: 17.85 pt

Deleted: 08

Deleted: 27

## 9 Specifying the Java Implementation Type in an Assembly

651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674

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

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

The implementation.java element has the following attributes:

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

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

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

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

Formatted: Indent: Before: 0 pt

Formatted: Indent: Before: 17.85 pt

Formatted: Indent: Before: 125.85 pt, First line: 18.15 pt

Formatted: Indent: Before: 0 pt

Formatted: Indent: Before: 17.85 pt, Hanging: 17.85 pt, Tabs: 18 pt, List tab + Not at 36 pt

Formatted: Indent: Before: 0 pt

Deleted: 08

Deleted: 27

675

## 10 Java Packaging and Deployment Model

676 | The SCA Assembly [Model](#) Specification [ASSEMBLY] describes the basic packaging model for SCA  
677 contributions in the chapter on Packaging and Deployment. This specification defines extensions to the  
678 basic model for SCA contributions that contain Java component implementations.

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

### 684 10.1 Contribution Metadata Extensions

685 SCA contributions can be self contained such that all the code and metadata needed to execute the  
686 components defined by the contribution is contained within the contribution. However, in larger  
687 projects, there is often a need to share artifacts across contributions. This is accomplished through  
688 the use of the import and export extension points as defined in the sca-contribution.xml document.  
689 An SCA contribution that needs to use a Java class from another contribution can declare the  
690 dependency via an <import.java/> extension element, contained within a <contribution/> element, as  
691 defined below:

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

Formatted: Indent: Before: 17.85 pt

693

694 The import.java element has the following attributes:

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

698

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

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

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

704

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

706

707 [1.2,1.3] - enclosing square brackets - inclusive range meaning any version in the range from  
708 the lowest to the highest, including the lowest and the highest

709 (1.3.1,2.4.1) - enclosing round brackets - exclusive range meaning any version in the range  
710 from the lowest to the highest but not including the lowest or the highest.

711 1.4.1 - no enclosing brackets - implies any version at or later than the specified version

712 number is acceptable - equivalent to [1.4.1, infinity)

713

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

716

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

719 Each Java package that is imported into the contribution MUST be included in one and only one

720 import.java element. [JCI100001] Multiple packages can be imported, either through specifying

721 multiple packages in the @package attribute or through the presence of multiple import.java

722 elements.

723 The SCA runtime MUST ensure that the package used to satisfy an import matches the package name,

724 the version number or version number range and (if present) the location specified on the import.java

725 element [JCI100002]

Deleted: 08

Deleted: 27

726 An SCA contribution that wants to allow a Java package to be used by another contribution can  
727 declare the exposure via an <export.java/> extension element as defined below:

728 | <export.java package="xs:string" />

Formatted: Indent: Before: 17.85  
pt

729

730 The export.java element has the following attributes:

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

Deleted: 0

734 The package can have a **version number** appended to it, separated from the package name  
735 by a semicolon ";" followed by the text "version=" and the version number:  
736 package="com.acme.package1;version=1.4.1"

737

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

741

742 The uses directive indicates that the SCA runtime MUST ensure that any SCA contribution that  
743 imports this package from this exporting contribution also imports the same version as is used by  
744 this exporting contribution of any of the packages contained in the uses directive. [JC1100003]

745 Typically, the packages in the uses directive are packages used in the interface to the package  
746 being exported (eg as parameters or as classes/interfaces that are extended by the exported  
747 package). Example:

748

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

750

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

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

754 Each Java package that is exported from the contribution MUST be included in one and only one  
755 export.java element. [JC1100004] Multiple packages can be exported, either through specifying  
756 multiple packages in the @package attribute or through the presence of multiple export.java  
757 elements.

758 For example, a contribution that wants to:

- 759 • use classes from the *some.package* package from another contribution (any version)
- 760 • use classes of the *some.other.package* package from another contribution, at exactly version  
761 2.0.0
- 762 • expose the *my.package* package from its own contribution, with version set to 1.0.0

763 would specify an sca-contribution.xml file as follows:

764

765 | <?xml version="1.0" encoding="UTF-8"?>  
766 | <contribution xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200903>  
767 | ...  
768 | <import.java package="some.package" />  
769 | <import.java package="some.other.package;version=[2.0.0]" />  
770 | <export.java package="my.package;version=1.0.0" />  
771 | </contribution>

Formatted: Indent: Before: 17.85  
pt

Deleted: "

Deleted: "

Deleted: "

Deleted: "

Deleted: "

Deleted: "

772

773 A Java package that is specified on an export element MUST be contained within the contribution  
774 containing the export element. [JC1100007]

Deleted: 08

Deleted: 27

775

## 776 10.2 Java Artifact Resolution

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

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

Formatted: Indent: Before: 17.85 pt, Hanging: 18.15 pt

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

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

Formatted: Indent: Before: 36 pt, Hanging: 17.85 pt

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

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

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

Formatted: Indent: Before: 21.6 pt

792 [JCI100008]

Deleted: I

## 793 10.3 Class Loader Model

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

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

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

801 evaluates to true.

Deleted: ;

Formatted: Indent: Before: 17.85 pt, First line: 0 pt

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

804

Deleted: 08

Deleted: 27



## 805 11 Conformance

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

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

### 813 11.1 SCA Java Component Implementation Composite Document

814 An SCA Java Component Implementation Composite Document is an SCA Composite Document, as  
815 defined by the SCA Assembly [Model Specification Section 13.1 \[ASSEMBLY\]](#), that uses the  
816 <implementation.java> element. Such an SCA Java Component Implementation Composite Document  
817 MUST be a conformant SCA Composite Document, as defined by [ASSEMBLY], and MUST comply with  
818 the requirements specified in Section 9 of this specification.

Deleted: s

### 819 11.2 SCA Java Component Implementation Contribution Document

820 An SCA Java Component Implementation Contribution Document is an SCA Contribution Document, as  
821 defined by the SCA Assembly [Model specification Section 13.1 \[ASSEMBLY\]](#), that uses the contribution  
822 metadata extensions defined in Section 10. Such an SCA Java Component Implementation  
823 Contribution document MUST be a conformant SCA Contribution Document, as defined by  
824 [ASSEMBLY], and MUST comply with the requirements specified in Section 10 of this specification.

### 825 11.3 SCA Runtime

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

- 827 1. The implementation MUST meet all the conformance requirements defined by the SCA  
828 Assembly Model Specification [ASSEMBLY].
- 830 2. The implementation MUST reject an SCA Java Composite Document that does not conform to  
831 the sca-implementation-java.xsd schema.
- 832 3. The implementation MUST reject an SCA Java Contribution Document that does not conform to  
833 the sca-contribution-java.xsd schema.
- 834 4. The implementation MUST meet all the conformance requirements, specified in 'Section 11  
835 Conformance', from the SCA Java Common Annotations and APIs Specification [JAVACAA].
- 836 5. This specification permits an implementation class to use any and all the APIs and annotations  
837 defined in the Java Common Annotations and APIs Specification [JAVACAA], therefore the  
838 implementation MUST comply with all the statements in Appendix B: Conformance Items of  
839 [JAVACAA], notably all mandatory statements have to be implemented.
- 840 6. The implementation MUST comply with all statements related to an SCA Runtime, specified in  
841 'Appendix B: Conformance Items' of this specification, notably all mandatory statements have  
842 to be implemented.

Deleted: 1.

Formatted: Indent: Before: 17.85 pt, Hanging: 17.85 pt, Space After: 6 pt, Numbered + Level: 1 + Numbering Style: 1, 2, 3, ... + Start at: 1 + Alignment: Left + Aligned at: 39.6 pt + Tab after: 0 pt + Indent at: 57.6 pt, Tabs: Not at 45.8 pt + 91.6 pt + 137.4 pt + 183.2 pt + 229 pt + 274.8 pt + 320.6 pt + 366.4 pt + 412.2 pt + 458 pt + 503.8 pt + 549.6 pt + 595.4 pt + 641.2 pt + 687 pt + 732.8 pt

Deleted: 2.

Deleted: 3.

Deleted: 4.

Deleted: 5.

Deleted: 6.

Deleted: X

Deleted: 08

Deleted: 27

844

## A. XML Schemas

845

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

846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright(C) OASIS(R) 2005,2009. 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/200903"
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
  elementFormDefault="qualified">

  <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>

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

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

</schema>

```

Formatted: Indent: Before: 17.85 pt

878

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

879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright(C) OASIS(R) 2005,2009. 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/200903"
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
  elementFormDefault="qualified">

  <include schemaLocation="sca-core-1.1-cd03.xsd"/>

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

```

Formatted: Indent: Before: 17.85 pt

Deleted: 08  
Deleted: 27

895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906

```
<sequence>  
  <any namespace="##other" processContents="lax"  
    minOccurs="0" maxOccurs="unbounded"/>  
</sequence>  
  <attribute name="class" type="NCName" use="required"/>  
  <anyAttribute namespace="##other" processContents="lax"/>  
</extension>  
</complexContent>  
</complexType>  
</schema>
```

Deleted: 1

Deleted: 08  
Deleted: 27

907

## B. Conformance Items

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

910

Conformance ID	Description
[JCI20001]	The services provided by a Java-based implementation MUST have an interface defined in one of the following ways: <ul style="list-style-type: none"> <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>
[JCI20002]	Java implementation classes MUST implement all the operations defined by the service interface.
[JCI50001]	A Java implementation class MUST provide a public or protected constructor that can be used by the SCA runtime to create the implementation instance.
[JCI50002]	The @Constructor annotation MUST only be specified on one constructor; the SCA container MUST raise an error if multiple constructors are annotated with @Constructor.
[JCI50003]	Cyclic references between components MUST be handled by the SCA runtime in one of two ways: <ul style="list-style-type: none"> <li>If any reference in the cycle is optional, then the container can inject a null value during construction, followed by injection of a reference to the target before invoking any service.</li> <li>The container can inject a proxy to the target service; invocation of methods on the proxy can result in a ServiceUnavailableException</li> </ul>
[JCI50004]	The constructor to use for the creation of an implementation instance MUST be selected by the SCA runtime using the sequence: <ol style="list-style-type: none"> <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>
[JCI50005]	The SCA runtime MUST raise an error if there are multiple constructors that are not annotated with @Constructor and have a non-empty parameter list with all parameters annotated with either @Property or @Reference.
[JCI60001]	The SCA runtime MUST support the STATELESS and COMPOSITE implementation scopes.
[JCI80001]	An SCA runtime MUST introspect the componentType of a Java implementation class following the rules defined in the section "Component Type of a Java Implementation".
[JCI80002]	If a Java implementation class, with or without @Property and @Reference annotations, has more than one setter method with the same JavaBeans property name [JAVABEANS] corresponding to the setter method name, then if more than

Formatted Table

Deleted: [JCI20001] ... [66]

Deleted: ¶

Formatted: Bulleted + Level: 1 + Aligned at: 18 pt + Tab after: 0 pt + Indent at: 36 pt

Formatted: English U.S.

Deleted: [JCI50003]

Deleted: ¶

Formatted: Bulleted + Level: 1 + Aligned at: 18 pt + Tab after: 0 pt + Indent at: 36 pt

Formatted: Indent: Before: 17.85 pt, Hanging: 18.15 pt

Deleted: [JCI50004] ... [67]

Deleted: 08

Deleted: 27

	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.
[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"> <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.
[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.

Formatted: Indent: Before: 17.85 pt, Hanging: 18.15 pt

Deleted:

Deleted:

Formatted: Indent: Before: 36 pt, Hanging: 17.85 pt

Deleted:

Deleted:

Deleted: j

Formatted: Indent: Before: 17.85 pt, Hanging: 18.15 pt

Deleted:

Deleted: 08

Deleted: 27

[JC1100011]

The SCA runtime MUST ensure that Java classes that are imported into a contribution are loaded by the exporting contribution's class loader

911

Deleted: 08

Deleted: 27

## 912 C. Acknowledgements

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

### 915 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 Combella	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: 08

Deleted: 27

Scott Vorthmann  
Feng Wang  
Robin Yang

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

916


Deleted: 08

Deleted: 27



## D. Non-Normative Text

---



Deleted: 08

Deleted: 27

918 **E. Revision History**

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

920

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
<a href="#">wd09</a>	<a href="#">2009-04-29</a>	<a href="#">David Booz</a>	<a href="#">Editorial fixes throughout (capitalization, quotes, fonts, spec references, etc.)</a>
<a href="#">wd10</a>	<a href="#">2009-04-30</a>	<a href="#">David Booz</a>	<a href="#">Editorial fixes, indentation, etc.</a>

921

922

Deleted: 08

Deleted: 27

Page 11: [1] Formatted Simon Nash 5/2/2009 9:02 PM

Normal, Indent: Before: 17.85 pt, Hanging: 18.15 pt

Page 11: [2] Deleted Simon Nash 5/2/2009 5:43 PM

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

1. A declared constructor annotated with a @Constructor annotation.
2. A declared constructor, all of whose parameters are annotated with either @Property or @Reference.
3. A no-argument constructor. The constructor to use for the creation of an implementation instance MUST be selected by the SCA runtime using the sequence:
  1. A declared constructor annotated with a @Constructor annotation.
  2. A declared constructor, all of whose parameters are annotated with either @Property or @Reference.
  3. A no-argument constructor.

Page 11: [3] Formatted Dave Booz 4/30/2009 10:30 AM

Indent: Before: 0 pt

Page 11: [4] Formatted Simon Nash 5/2/2009 4:52 PM

Bulleted + Level: 1 + Aligned at: 18 pt + Tab after: 0 pt + Indent at: 36 pt

Page 11: [5] Formatted Simon Nash 5/2/2009 6:08 PM

Normal, Bulleted + Level: 1 + Aligned at: 18 pt + Tab after: 0 pt + Indent at: 36 pt

Page 11: [6] Deleted Simon Nash 5/2/2009 6:07 PM

Cyclic references between components MUST be handled by the SCA runtime in one of two ways:

- If any reference in the cycle is optional, then the container can inject a null value during construction, followed by injection of a reference to the target before invoking any service.
- The container can inject a proxy to the target service; invocation of methods on the proxy can result in a ServiceUnavailableException. Cyclic references between components MUST be handled by the SCA runtime in one of two ways:
  - If any reference in the cycle is optional, then the container can inject a null value during construction, followed by injection of a reference to the target before invoking any service.
  - The container can inject a proxy to the target service; invocation of methods on the proxy can result in a ServiceUnavailableException.

Page 11: [7] Formatted Dave Booz 4/30/2009 10:30 AM

Indent: Before: 0 pt

Page 11: [8] Formatted Simon Nash 5/2/2009 9:51 PM

Font: 10 pt

Page 11: [9] Formatted Simon Nash 5/2/2009 9:53 PM

Indent: Before: 17.85 pt, Space After: 0 pt

Page 11: [10] Formatted Simon Nash 5/2/2009 9:51 PM

Font: 10 pt

Page 11: [10] Formatted Simon Nash 5/2/2009 9:51 PM

Font: (Default) Courier New, 10 pt, Complex Script Font: Courier New

Page 11: [11] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: (Default) Courier New, 10 pt, Complex Script Font: Courier New

Page 11: [11] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [11] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: (Default) Courier New, 10 pt, Complex Script Font: Courier New

Page 11: [11] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [12] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [12] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [13] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [13] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [13] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [13] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [14] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [14] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [14] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [14] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [14] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [14] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [14] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [14] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [14] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [15] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [15] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [16] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [17] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [17] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [18] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [18] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [19] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [19] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [19] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [20] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [20] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [21] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [21] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [22] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [22] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [23] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [23] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [24] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [24] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [24] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [24] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [25] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [25] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [25] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [25] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [25] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [25] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [25] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [26] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [26] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [26] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [26] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [26] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [26] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [26] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [26] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [27] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [27] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [27] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [27] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [28] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [28] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [29] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [30] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [31] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [31] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [32] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [33] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [33] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [34] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [34] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [34] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [35] Deleted	Dave Booz	4/30/2009 10:47 AM
-----------------------	-----------	--------------------

Page 11: [36] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [36] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [36] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [37] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [37] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [37] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 1: [38] Deleted	Dave Booz	4/30/2009 10:33 AM
----------------------	-----------	--------------------

08

Page 1: [38] Deleted	Dave Booz	4/30/2009 10:33 AM
----------------------	-----------	--------------------

27

Page 11: [39] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt

Page 11: [39] Formatted	Simon Nash	5/2/2009 9:51 PM
-------------------------	------------	------------------

Font: 10 pt





Page 11: [44] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [45] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [45] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [45] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [45] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [45] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [46] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [46] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [46] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [46] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [47] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [47] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [47] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [47] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [48] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [48] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [49] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [50] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [50] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [50] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [50] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM

Page 11: [51] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [51] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [52] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [52] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [52] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [52] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [53] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [53] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [54] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [54] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [54] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [54] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [55] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [55] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [55] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [55] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [56] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [56] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [56] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [56] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [57] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM

Page 11: [57] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [57] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [58] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [59] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [59] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [59] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [59] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [59] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [59] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [60] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [60] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [61] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 11: [61] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 12: [61] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 12: [62] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 12: [62] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 12: [62] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 12: [62] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 12: [62] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 12: [63] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM
Page 12: [63] Formatted Font: 10 pt	Simon Nash	5/2/2009 9:51 PM

Page 12: [63] Formatted Simon Nash 5/2/2009 9:51 PM

Font: 10 pt

Page 12: [63] Formatted Simon Nash 5/2/2009 9:51 PM

Font: 10 pt

Page 12: [63] Formatted Simon Nash 5/2/2009 9:51 PM

Font: 10 pt

Page 12: [64] Formatted Simon Nash 5/2/2009 9:51 PM

Font: 10 pt

Page 12: [64] Formatted Simon Nash 5/2/2009 9:51 PM

Font: 10 pt

Page 1: [65] Deleted Dave Booz 4/30/2009 10:33 AM

08

Page 1: [65] Deleted Dave Booz 4/30/2009 10:33 AM

27

Page 28: [66] Deleted Simon Nash 5/2/2009 5:20 PM

<p>[JCI20001]</p>	<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"><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>
-------------------	--

Page 28: [67] Deleted Simon Nash 5/2/2009 5:11 PM

<p>[JCI50004]</p>	<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"><li>1. A declared constructor annotated with a @Constructor annotation.</li><li>2. A declared constructor, all of whose parameters are annotated with either @Property or @Reference.</li><li>3. A no-argument constructor.</li></ol>
-------------------	--