# OASIS 🕅

# Service Component Architecture Java Component Implementation Specification Version 1.1

### Working Draft 10 Deleted: 8 30th April 2009 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 Combellack, 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 Formatted: Bullets and Numbering Version 1.1 Declared XML Namespace(s): http://docs.oasis-open.org/ns/opencsa/sca/200903 Deleted: 08 Deleted: 27

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved. 30th April 2009 Page 1 of 34 Deleted: +simon

### 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.

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-i/.

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.oasisopen.org/committees/sca-j/. Deleted:

Deleted: 08

Deleted: 27

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved. 30th April 2009 Page 2 of 34

# 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

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved.

# **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
1(	) Java Packaging and Deployment Model	22
	10.1 Contribution Metadata Extensions	22
	10.2 Java Artifact Resolution	24
	10.3 Class loader Model	24
1	Conformance	25
	11.1 SCA Java Component Implementation Composite Document	25
	11.2 SCA Java Component Implementation Contribution Document	
	11.3 SCA Runtime	25
А	XML Schemas	26
	A.1 sca-contribution-java.xsd	26
	A.2 sca-implementation-java.xsd	26
В	Conformance Items	28
С	Acknowledgements	31
D	Non-Normative Text	33
Е	Revision History	

Deleted: 08

Deleted: 27

sca-javaci-1.1-spec-wd<u>10</u> Copyright © OASIS® 2005,2009. All Rights Reserved. 30th April 2009 Page 4 of 34

#### 1 Introduction 1

2 This specification extends the SCA Assembly Model [ASSEMBLY] by defining how a Java class provides 3 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. 4

5 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 6 7 are defined in the former unless otherwise specified. Moreover, the semantics defined in the Common 8 Annotations and APIs specification are normative.

9 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 10 11 implementation type

#### 1.1 Terminology 12

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD 13

NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **[RFC2119]**. 14

### 15

#### **1.2 Normative References** 16

17 18 19	[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.		
20	[ASSEMBLY]	SCA Assembly Model Specification Version 1.1,	+	Formatted: English (United States)
21		http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec.pdf		Deleted: http://www.oasis- open.org/committees/download.php/31
22 23	[POLICY]	SCA Policy Framework Specification Version 1.1,		722/sca-assembly-1.1-spec-cd03.pdf
24 25		http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.pdf		Deleted: http://www.oasis- open.org/committees/download.php/31
26	[JAVACAA]	SCA Java Common Annotations and APIs Specification Version 1.1,		608/sca-policy-1.1-spec-cd02.pdf
27 28		http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd01.pdf		<b>Comment [DAB1]:</b> Update to CD03/PRD link at the appropriate
29 30	[WSDL]	WSDL Specification, WSDL 1.1: http://www.w3.org/TR/wsdl		time.
31	[OSGi Core]	OSGI Service Platform Core Specification, Version 4.0.1		
32 33		http://www.osgi.org/download/r4v41/r4.core.pdf		
34 35	[JAVABEANS]	JavaBeans 1.01 Specification, http://java.sun.com/javase/technologies/desktop/javabeans/api/		

#### **1.3 Non-Normative References** 36

TBD 37

TBD

Deleted: 08

Deleted:

Deleted: in [JAVACAA]

Deleted: 27

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved. 30th April 2009 Page 5 of 34

## 46 2 Service

```
A component implementation based on a Java class can provide one or more services
47
                                                                                                                       Formatted: Indent: Left: 0 cm
48
       The services provided by a Java-based implementation MUST have an interface defined in one of the
                                                                                                                       Deleted:
49
      following ways:
50

    A Java interface

                                                                                                                       Formatted: Bulleted + Level: 1 +
                                                                                                                       Aligned at: 0.63 cm + Indent at: 1.27
51
              A Java class
                                                                                                                       cm
              A Java interface generated from a Web Services Description Language [WSDL] (WSDL)
52
                                                                                                                       Formatted: English (United States)
53
              portType.
54
      [JCI20001]
                                                                                                                       Deleted: ¶
                                                                                                                       The services provided by a Java-
       Java implementation classes MUST implement all the operations defined by the service interface.
55
                                                                                                                       based implementation MUST have
      [JCI20002] If the service interface is defined by a Java interface, the Java-based component can
56
                                                                                                                       an interface defined in one of the
57
      either implement that Java interface, or implement all the operations of the interface.
                                                                                                                       following ways:¶

    A Java interface

58
      Java interfaces generated from WSDL portTypes are remotable, see the WSDL to Java and Java to

    A Java class

59
      WSDL section of the SCA Java Common Annotations and APIs Specification [JAVACAA] for details.

    A Java interface generated from

                                                                                                                       a Web Services Description
60
      A Java implementation type can specify the services it provides explicitly through the use of the
                                                                                                                       Language [WSDL] (WSDL)
      @Service annotation. In certain cases as defined below, the use of the @Service annotation is not
61
                                                                                                                       portType. The services provided by
      necessary and the services a Java implementation type offers can be inferred from the implementation
62
                                                                                                                       a Java-based implementation
      class itself.
63
                                                                                                                       MUST have an interface defined in
                                                                                                                       one of the following ways:¶
                                                                                                                       • A Java interface
      2.1 Use of @Service
64

    A Java class

    A Java interface generated from

      Service interfaces can be specified as a Java interface. A Java class, which is a component
65
                                                                                                                       a Web Services Description
      implementation, can offer a service by implementing a Java interface specifying the service contract.
66
                                                                                                                       Language [WSDL] (WSDL)
67
      As a Java class can implement multiple interfaces, some of which might not define SCA services, the
                                                                                                                       portType. ¶
      @Service annotation can be used to indicate the services provided by the implementation and their
68
                                                                                                                       Formatted: Indent: Left: 0 cm
69
      corresponding Java interface definitions.
                                                                                                                       Formatted: English (United States)
70
      The following is an example of a Java service interface and a Java implementation, which provides a
                                                                                                                       Formatted: Indent: Left: 0 cm
71
      service using that interface:
                                                                                                                       Deleted: WSDL 2 Java and Java 2
72
      Interface:
                                                                                                                       WSDL s
73
          package services.hello;
                                                                                                                       Formatted: Indent: Left: 0 cm
74
                                                                                                                       Deleted:
75
          public interface HelloService {
76
77
              String hello(String message);
                                                                                                                       Formatted: Indent: Left: 0.63 cm
78
          }
79
                                                                                                                       Formatted: Indent: Left: 0 cm
80
      Implementation class:
81
          @Service(HelloService.class)
          public class HelloServiceImpl implements HelloService {
82
83
84
              public String hello(String message) {
                                                                                                                       Deleted:
85
                                                                                                                       Formatted: Indent: Left: 1.27 cm,
86
                                                                                                                       First line: 0.63 cm
87
          }
88
                                                                                                                       Formatted: Indent: Left: 0.63 cm
                                                                                                                       Formatted: Indent: Left: 0 cm
89
      The XML representation of the component type for this implementation is shown below for illustrative
90
      purposes. There is no need to author the component type as it is introspected from the Java class.
                                                                                                                       Deleted: 08
                                                                                                                       Deleted: 27
      sca-javaci-1.1-spec-wd10,
                                                                                                   30th April 2009
      Copyright © OASIS® 2005,2009. All Rights Reserved.
                                                                                                     Page 6 of 34
```

```
116
117
          <?xml version="1.0" encoding="UTF-8"?>
118
          <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
119
120
              <service name="HelloService">
                 <interface.java interface="services.hello.HelloService"/>
121
                                                                                                            Deleted:
122
              </service>
                                                                                                            Formatted: Indent: Left: 1.27 cm,
123
                                                                                                            First line: 0.63 cm
124
          </componentType>
                                                                                                            Formatted: Indent: Left: 0.63 cm
125
                                                                                                            Formatted: Indent: Left: 0 cm
126
      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
127
128
      explicitly declare the implementation class defines the service offered by the implementation. In this
129
      case, a component will only offer services declared by @Service. The following illustrates this:
130
131
          package services.hello;
132
                                                                                                            Formatted: Indent: Left: 0 cm
133
          @Service(HelloServiceImpl.class)
134
          public class HelloServiceImpl implements AnotherInterface {
135
136
              public String hello(String message) {
137
                                                                                                            Deleted:
                 ••••
138
                                                                                                            Formatted: Indent: Left: 1.27 cm,
139
                                                                                                            First line: 0.63 cm
140
          }
                                                                                                            Formatted: Indent: Left: 0.63 cm
141
                                                                                                            Formatted: Indent: Left: 0 cm
142
      In the above example, HelloServiceImpl offers one service as defined by the public methods of the
143
       implementation class. The interface AnotherInterface in this case does not specify a service offered by
                                                                                                            Formatted: Indent: Left: 0 cm
       the component. The following is an XML representation of the introspected component type:
144
145
          <?xml version="1.0" encoding="UTF-8"?>
146
          <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
147
148
                                                                                                            Formatted: Indent: Left: 0 cm
              <service name="HelloServiceImpl">
149
                 <interface.java interface="services.hello.HelloServiceImpl"/>
                                                                                                            Deleted:
150
              </service>
                                                                                                            Formatted: Indent: Left: 1.27 cm,
151
                                                                                                            First line: 0.63 cm
152
          </componentType>
153
                                                                                                            Formatted: Indent: Left: 0 cm
                                                                                                            Formatted: Indent: Left: 0.63 cm
154
      The @Service annotation can be used to specify multiple services offered by an implementation as in
155
      the following example:
                                                                                                            Formatted: Indent: Left: 0 cm
                                                                                                            Deleted:
156
                                                                                                            Deleted:
157
          @Service(interfaces={HelloService.class, AnotherInterface.class})
158
          public class HelloServiceImpl implements HelloService, AnotherInterface
                                                                                                            Formatted: Indent: Left: 1.27 cm,
159
          {
                                                                                                            First line: 0.63 cm
160
                                                                                                            Deleted:
              public String hello(String message) {
161
                                                                                                            Formatted: Indent: First line: 1.27
162
                      -----
                                                                                                            cm
163
164
                                                                                                            Formatted: Indent: Left: 0 cm
165
          }
                                                                                                            Formatted: Indent: Left: 0.63 cm
166
                                                                                                            Formatted: Indent: Left: 0 cm
167
      The following snippet shows the introspected component type for this implementation.
                                                                                                            Deleted: 08
168
          <?xml version="1.0" encoding="UTF-8"?>
                                                                                                            Deleted: 27
      sca-javaci-1.1-spec-wd10
                                                                                          30th April 2009
       Copyright © OASIS® 2005,2009. All Rights Reserved.
                                                                                            Page 7 of 34
```

175	<componenttype xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"></componenttype>	<b>.</b>	Formatted: Indent: First line: 0.63
176			cm
177	<pre><service name="HelloService"></service></pre>	4	Formatted: Indent: Left: 0 cm
178 179	<pre><interface.java interface="services.hello.HelloService"></interface.java> </pre>		Deleted:
180 181	<pre><service name="AnotherService"></service></pre>		Formatted: Indent: Left: 1.27 cm, First line: 0.63 cm
182		•	Formatted: Indent: Left: 0 cm
183 184			Deleted:
			Formatted: Indent: Left: 1.27 cm, First line: 0.63 cm
185	2.2 Local and Remotable <u>S</u> ervices	· - (``\	Formatted: Indent: Left: 0 cm
186	A Java service contract defined by an interface or implementation class uses the @Remotable	<ul> <li>▲</li> <li></li> <li>&lt;</li></ul>	Formatted: Indent: Left: 0.63 cm
187 188	annotation to declare that the service follows the semantics of remotable services as defined by the <u>SCA Assembly Model Specification [ASSEMBLY]</u> . The following example demonstrates the use of the		Deleted: s
189	@Remotable annotation:		Formatted: Indent: Left: 0 cm
190	<pre>package services.hello;</pre>		Deleted: SCA Assembly
191			Specification
192 193	<pre>@Remotable public interface HelloService {</pre>		
194			
195 196	String hello(String message);	<b>4</b>	Formatted: Indent: Left: 0.63 cm
190	}	4	Formatted: Indent: Left: 0.63 cm
198	Unless annotated with a @Remotable annotation, a service defined by a Java interface or a Java		Formatted: Indent: Left: 0 cm
199 200	implementation class is inferred to be a local service as defined by the SCA Assembly Model Specification [ASSEMBLY].		
201	An implementation class can provide hints to the SCA runtime about whether it can achieve pass-by-		Deleted
201 202	An implementation class can provide hints to the SCA runtime about whether it can achieve pass-by- value semantics without making a copy by using the @AllowsPassByReference annotation.		Deleted:
	value semantics without making a copy by using the @AllowsPassByReference annotation.		Deleted: s
202 203	value semantics without making a copy by using the @AllowsPassByReference annotation. 2.3 Introspecting Services Offered by a Java Implementation		Deleted: s Deleted: o
202	value semantics without making a copy by using the @AllowsPassByReference annotation.		Deleted: s Deleted: o Deleted: i
202 203 204	value semantics without making a copy by using the @AllowsPassByReference annotation. <b>2.3 Introspecting Services Offered by a Java Implementation</b> The services offered by a Java implementation class are determined through introspection, as defined		Deleted: s Deleted: o
202 203 204 205 206 207 208 209 210	<ul> <li>value semantics without making a copy by using the @AllowsPassByReference annotation.</li> <li><b>2.3 Introspecting Services Offered by a Java Implementation</b></li> <li>The services offered by a Java implementation class are determined through introspection, as defined in the section "Component Type of a Java Implementation".</li> <li>If the interfaces of the SCA services are not specified with the @Service annotation on the implementation class, it is assumed that all implemented interfaces that have been annotated as @Remotable are the service interfaces provided by the component. If an implementation class has only implemented interfaces that are not annotated with a @Remotable annotation, the class is considered to implement a single <i>local</i> service whose type is defined by the class (note that local</li> </ul>		Deleted: s Deleted: o Deleted: i
202 203 204 205 206 207 208 209 210 211	<ul> <li>value semantics without making a copy by using the @AllowsPassByReference annotation.</li> <li><b>2.3 Introspecting Services Offered by a Java Implementation</b></li> <li>The services offered by a Java implementation class are determined through introspection, as defined in the section "Component Type of a Java Implementation".</li> <li>If the interfaces of the SCA services are not specified with the @Service annotation on the implementation class, it is assumed that all implemented interfaces that have been annotated as @Remotable are the service interfaces provided by the component. If an implementation class has only implemented interfaces that are not annotated with a @Remotable annotation, the class is considered to implement a single <i>local</i> service whose type is defined by the class (note that local services can be typed using either Java interfaces or classes).</li> </ul>		Deleted: s Deleted: o Deleted: i
202 203 204 205 206 207 208 209 210 211 212 213 214 215	<ul> <li>value semantics without making a copy by using the @AllowsPassByReference annotation.</li> <li><b>2.3 Introspecting Services Offered by a Java Implementation</b></li> <li>The services offered by a Java implementation class are determined through introspection, as defined in the section "Component Type of a Java Implementation".</li> <li>If the interfaces of the SCA services are not specified with the @Service annotation on the implementation class, it is assumed that all implemented interfaces that have been annotated as @Remotable are the service interfaces provided by the component. If an implementation class has only implemented interfaces that are not annotated with a @Remotable annotation, the class is considered to implement a single <i>local</i> service whose type is defined by the class (note that local services can be typed using either Java interfaces or classes).</li> <li><b>2.4 Non-Blocking Service Operations</b></li> <li>Service operations defined by a Java interface or by a Java implementation class can use the @OneWay annotation to declare that the SCA runtime needs to honor non-blocking semantics as defined by the SCA Assembly Model_Specification [ASSEMBLY] when a client invokes the service</li> </ul>		Deleted: s Deleted: o Deleted: i Formatted: Indent: Left: 0 cm
202 203 204 205 206 207 208 209 210 211 212 213 214 215 216	<ul> <li>value semantics without making a copy by using the @AllowsPassByReference annotation.</li> <li><b>2.3 Introspecting Services Offered by a Java Implementation</b></li> <li>The services offered by a Java implementation class are determined through introspection, as defined in the section "Component Type of a Java Implementation".</li> <li>If the interfaces of the SCA services are not specified with the @Service annotation on the implementation class, it is assumed that all implemented interfaces that have been annotated as @Remotable are the service interfaces provided by the component. If an implementation class has only implemented interfaces that are not annotated with a @Remotable annotation, the class is considered to implement a single <i>local</i> service whose type is defined by the class (note that local services can be typed using either Java interfaces or classes).</li> <li><b>2.4 Non-Blocking Service Operations</b></li> <li>Service operations defined by a Java interface or by a Java implementation class can use the @OneWay annotation to declare that the SCA runtime needs to honor non-blocking semantics as defined by the SCA Assembly Model_Specification [ASSEMBLY] when a client invokes the service operation.</li> </ul>		Deleted: s Deleted: o Deleted: i Formatted: Indent: Left: 0 cm Formatted: Indent: Left: 0 cm
202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220	<ul> <li>value semantics without making a copy by using the @AllowsPassByReference annotation.</li> <li><b>2.3 Introspecting Services Offered by a Java Implementation</b></li> <li>The services offered by a Java implementation class are determined through introspection, as defined in the section "Component Type of a Java Implementation".</li> <li>If the interfaces of the SCA services are not specified with the @Service annotation on the implementation class, it is assumed that all implemented interfaces that have been annotated as @Remotable are the service interfaces provided by the component. If an implementation class has only implemented interfaces that are not annotated with a @Remotable annotation, the class is considered to implement a single <i>local</i> service whose type is defined by the class (note that local services can be typed using either Java interfaces or classes).</li> <li><b>2.4 Non-Blocking Service Operations</b></li> <li>Service operations defined by a Java interface or by a Java implementation class can use the @OneWay annotation to declare that the SCA runtime needs to honor non-blocking semantics as defined by the SCA Assembly <u>Model</u> Specification [ASSEMBLY] when a client invokes the service operation.</li> <li><b>2.5 Callback Services</b></li> <li>A callback interface can be declared by using the @Callback annotation on the service interface or Java implementation class as described in the Java Common Annotations and APIs Specification [JAVACAA]. Alternatively, the @callbackInterface attribute of the <interface.java></interface.java> element can be</li> </ul>		Deleted: s Deleted: o Deleted: i Formatted: Indent: Left: 0 cm Formatted: Indent: Left: 0 cm Deleted: 08
202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220	<ul> <li>value semantics without making a copy by using the @AllowsPassByReference annotation.</li> <li><b>2.3 Introspecting Services Offered by a Java Implementation</b></li> <li>The services offered by a Java implementation class are determined through introspection, as defined in the section "Component Type of a Java Implementation".</li> <li>If the interfaces of the SCA services are not specified with the @Service annotation on the implementation class, it is assumed that all implemented interfaces that have been annotated as @Remotable are the service interfaces provided by the component. If an implementation class has only implemented interfaces that are not annotated with a @Remotable annotation, the class is considered to implement a single <i>local</i> service whose type is defined by the class (note that local services can be typed using either Java interfaces or classes).</li> <li><b>2.4 Non-Blocking Service Operations</b></li> <li>Service operations defined by a Java interface or by a Java implementation class can use the @OneWay annotation to declare that the SCA runtime needs to honor non-blocking semantics as defined by the SCA Assembly <u>Model</u> Specification [ASSEMBLY] when a client invokes the service operation.</li> <li><b>2.5 Callback Services</b></li> <li>A callback interface can be declared by using the @Callback annotation on the service interface or Java implementation class as described in the Java Common Annotations and APIs Specification [JAVACAA]. Alternatively, the @callbackInterface attribute of the <interface.java></interface.java> element can be</li> </ul>	•	Deleted: s Deleted: o Deleted: i Formatted: Indent: Left: 0 cm Formatted: Indent: Left: 0 cm

#### 3 References 231

232 A Java implementation class can obtain service references either through injection or through the 233 ComponentContext API as defined in the SCA Java Common Annotations and APIs Specification [JAVACAA]. When possible, the preferred mechanism for accessing references is through injection. 234 3.1 Reference Injection 235 236 A Java implementation type can explicitly specify its references through the use of the @Reference 237 annotation as in the following example: 238 239 public class ClientComponentImpl implements Client { private HelloService service; 240 241 242 @Reference public void setHelloService(HelloService service) { 243 244 this.service = service; 245 } 246 } 247 248 If @Reference marks a setter method, the SCA runtime provides the appropriate implementation of 249 the service reference contract as specified by the parameter type of the method. This is done by 250 invoking the setter method of an implementation instance of the Java class. When injection occurs is 251 defined by the scope of the implementation. However, injection always occurs before the first service 252 method is called. 253 If @Reference marks a field, the SCA runtime provides the appropriate implementation of the service 254 reference contract as specified by the field type. This is done by setting the field on an implementation 255 instance of the Java class. When injection occurs is defined by the scope of the implementation. 256 However, injection always occurs before the first service method is called. 257 If @Reference marks a parameter on a constructor, the SCA runtime provides the appropriate 258 implementation of the service reference contract as specified by the constructor parameter during 259 creation of an implementation instance of the Java class. **Deleted:** instantiation 260 Except for constructor parameters, references marked with the @Reference annotation can be Deleted: R declared with required=false, as defined by the Java Common Annotations and APIs Specification 261 262 [JAVACAA] - i.e., the reference multiplicity is 0..1 or 0..n, where the implementation is designed to cope with the reference not being wired to a target service. 263 264 In the case where a Java class contains no @Reference or @Property annotations, references are determined by introspecting the implementation class as described in the section "ComponentType of 265 266 an Implementation with no @Reference or @Property annotations ". 3.2 Dynamic Reference Access 267 268 As an alternative to reference injection, service references can be accessed dynamically through the 269 API methods ComponentContext.getService() and ComponentContext.getServiceReference() methods

270 as described in the Java Common Annotations and APIS Specification [JAVACAA]. Formatted: Indent: Left: 0 cm

Formatted: Indent: Left: 0 cm Formatted: Indent: Left: 0.63 cm Deleted: Deleted: Deleted: Deleted: Deleted: Formatted: Indent: Left: 1.27 cm. First line: 0.63 cm Formatted: Indent: Left: 0.63 cm Formatted: Indent: Left: 0 cm, First line: 0.63 cm

Formatted: Indent: Left: 0 cm

Formatted: Indent: Left: 0 cm

Deleted: 08 Deleted: 27

sca-javaci-1.1-spec-wd10 Copyright © OASIS® 2005,2009. All Rights Reserved. 30th April 2009 Page 9 of 34

#### **4** Properties 278

29

298

#### 4.1 Property Injection 279

280 Properties can be obtained either through injection or through the ComponentContext API as defined 281 in the SCA Java Common Annotations and APIs Specification [JAVACAA]. When possible, the preferred mechanism for accessing properties is through injection. 282

283 A Java implementation type can explicitly specify its properties through the use of the @Property annotation as in the following example: 284

<pre>public_class_ClientComponentImpl implements Client {     private int maxRetries;</pre>
@Property
<pre>public void setMaxRetries(int maxRetries) {</pre>
<pre>this.maxRetries = maxRetries;</pre>
}
}
If the @Property annotation marks a setter method, the SCA runtime provides the appropriate property value by invoking the setter method of an implementation instance of the Java class. When injection occurs is defined by the scope of the implementation. However, injection always occurs

If the @Property annotation marks a field, the SCA runtime provides the appropriate property value 299 300 by setting the value of the field of an implementation instance of the Java class. When injection occurs 301 is defined by the scope of the implementation. However, injection always occurs before the first 302 service method is called.

303 If the @Property annotation marks a parameter on a constructor, the SCA runtime provides the 304 appropriate property value during creation of an implementation instance of the Java class.

305 Except for constructor parameters, properties marked with the @Property annotation can be declared 306 with required=false as defined by the Java Common Annotations and APIs Specification [JAVACAA], 307 i.e., the property mustSupply attribute is false and where the implementation is designed to cope with 308 the component configuration not supplying a value for the property.

In the case where a Java class contains no @Reference or @Property annotations, properties are 309 310 determined by introspecting the implementation class as described in the section "ComponentType of 311 an Implementation with no @Reference or @Property annotations ".

#### 4.2 Dynamic Property Access 312

before the first service method is called.

313 As an alternative to property injection, properties can also be accessed dynamically through the

ComponentContext.getProperty() method as described in the Java Common Annotations and APIs 314 315 Specification [JAVACAA].

Formatted: Indent: Left: 0 cm

Formatted: Indent: Left: 0.63 cm
Deleted:
Formatted: Indent: Left: 1.27 cm, First line: 0.63 cm
Formatted: Indent: Left: 0.63 cm
Formatted: Indent: Left: 0.63 cm, First line: 0 cm
Formatted: Indent: Left: 0 cm

Deleted: instantiation

Deleted: P

Formatted: Indent: Left: 0 cm

Deleted: 08

Deleted: 27

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved. 30th April 2009 Page 10 of 34

	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 the
	nanner are set into the field or are passed to the setter method associated with the property or
	eference before any service method is invoked.
I	The constructor to use for the creation of an implementation instance MUST be selected by the SCA
	untime using the sequence:
	<ol> <li>A declared constructor annotated with a @Constructor annotation.</li> </ol>
	2. A declared constructor, all of whose parameters are annotated with either @Property or
	@Reference.
	3. A no-argument constructor.
,	JCI50004]
ļ	he @Constructor annotation MUST only be specified on one constructor; the SCA container MUST
	aise an error if multiple constructors are annotated with @Constructor. [JCI50002]
	he 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]
	he property or reference associated with each parameter of a constructor is identified through the resence 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:
	<ul> <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> </ul>
	<ul> <li>The container can inject a proxy to the target service; invocation of methods on the proxy can result in a ServiceUnavailableException</li> </ul>
	he following are examples of legal Java component constructor declarations:
	<pre>/** Constructor declared using @Constructor annotation */</pre>
	public class Impl1 {
	private String someProperty;
	@Constructor
	<pre>public Impl1( @Property("someProperty") String propval ) {}</pre>
	v
	$\checkmark$ ** Declared constructor unambiguously identifying all Property.
	* and Reference values */
	public class Impl2 {
	<pre>private String someProperty; private SomeService someReference;</pre>
	<pre>public Impl2( @Property("someProperty") String a,</pre>
	<pre>@Reference("someReference") SomeService b )</pre>
	•{}
	√
	/** Declared constructor unambiguously identifying all Property
	*, and Reference values plus an additional Property injected

-
Formatted: Normal, Indent: Left: 0.63 cm, Hanging: 0.64 cm
Deleted: The constructor to use [2]
Formatted: Normal
Formatted: Indent: Left: 0 cm
Formatted [3]
Deleted: ¶
Formatted: Indent: Left: 0 cm
Deleted:
Formatted: Font: 10 pt
Formatted [5]
Formatted [6]
Deleted:
Formatted [7]
Deleted:
Deleted:
Deleted:
Deleted:
Deleted:** Declared [[8]
Deleted: /
Formatted: Font: 10 pt
Deleted: *
Formatted: Font: 10 pt
Deleted:
Deleted:** Declared [9]
Deleted: /
Formatted: Font: 10 pt
Deleted: *
Formatted: Font: 10 pt
Deleted: ¶ [10]
Deleted: */
Deleted: . /
Formatted: Font: 10 pt
Deleted: *
Formatted: Font: 10 pt
Deleted: 08 , 3027

517	<pre>public class Impl3 {</pre>		Deleted:
518	<pre>private String someProperty;</pre>		Deleted:
519	<pre>private String anotherProperty;</pre>		Deleted.
520	<pre>private SomeService someReference;</pre>		Deleted:
521	public Impl3( @Property("someProperty") String a,	\```	Deleted:
522 523	<pre>@Reference("someReference") SomeService b)</pre>	\```	Deleted:
524	<pre>{} @Property</pre>	\``\.	
525	<pre>public void setAnotherProperty( String anotherProperty ) {}</pre>	\`\\	Deleted:
525 526	<pre>public void setAnotherproperty( string anotherproperty ) {}</pre>	(`\(``	Deleted:
527	۷		Deleted:
528	/** No-arg constructor */	· \	Deleted:
529 530	<pre>public class Impl4 {     @Property</pre>	= = = ¬(```, ``	Deleted:
531			
	public String someProperty;		Deleted:
532 533	<pre>@Reference public SomeService someReference;</pre>		Deleted:
534	public Impl4() {}	$===\sum_{i=1}^{N}\sum_{i=1}^{N}\sum_{i=1}^{N}$	Deleted:
535	√	,``,``	Deleted:
536 537	<pre>/** Unannotated implementation with no-arg constructor */</pre>		Deleted:
538	public class Imp15 {	\`\`\`\	Deleted:
539	public String someProperty;	···	
540	public SomeService someReference;		Deleted: .
541	public Imp15() {}	·····	Deleted:
542	*	$=$ $=$ $=$ $=$ $=$ $=$ $=$ $\frac{1}{2}$	Deleted:
		1 March	Deleted:

Deleted: .
Deleted: .
Deleted: .
Formatted: Font: 10 pt
Deleted: .
Formatted: Font: 10 pt
Deleted: .

Formatted: Font: 10 pt

Deleted:

Deleted: 08

Deleted: 27

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved. 30th April 2009 Page 12 of 34

567	6 Implementation Scopes and Lifecycle Callbacks		
568 569 570	The Java implementation type supports all of the scopes defined in the Java Common Annotations and API <u>S</u> Specification: STATELESS and COMPOSITE. The SCA runtime MUST support the STATELESS and COMPOSITE implementation scopes. [JCI60001]		
571	Implementations specify their scope through the use of the @Scope annotation as in:	<	Formatted: Indent: Left: 0 cm
572		•	Formatted: Indent: Left: 0.63 cm
573 574	<pre>@Scope("COMPOSITE") public class ClientComponentImpl implements Client {</pre>	7	Deleted:
575	//		Deleted: "
576 577	}		Deleted: "
577 578	When the @Scope annotation is not specified on an implementation class, its scope is defaulted to STATELESS.		Deleted:
579 580	A Java component implementation specifies init and destroy <u>methods</u> by using the @Init and @Destroy annotations respectively, as described in the Java Common Annotations and APIs	-	Formatted: Indent: Left: 0.63 cm, First line: 0 cm
581	specification [JAVACAA].		Formatted: Indent: Left: 0 cm
582	For example:		Deleted: callbacks
583 584	<pre>public class ClientComponentImpl implements Client {</pre>		Deleted:
585	@Init		Formatted: Indent: Left: 0.63 cm
586 587	<pre>public void init() {</pre>		Formatted: Indent: Left: 0.63 cm, First line: 0 cm
588 589	}		Formatted: Indent: Left: 0.63 cm, First line: 0.64 cm
590	@Destroy		Deleted:
591 592 593	<pre>public void destroy() {</pre>		Formatted: Indent: Left: 1.27 cm, First line: 0.63 cm
594 595	}		Formatted: Indent: Left: 0.63 cm, First line: 0 cm
		-1 $h$	Formatted: Indent: Left: 0.63 cm,

sca-javaci-1.1-spec-wd<u>10.</u> Copyright © OASIS® 2005,2009. All Rights Reserved. Deleted:

Deleted: 08

Formatted: Indent: Left: 1.27 cm, First line: 0.63 cm

Formatted: Indent: Left: 0.63 cm, First line: 0 cm

# 604 7 Accessing a Callback Service

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

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

- - Formatted: Indent: Left: 0 cm

Deleted: 08

Deleted: 27

30th April 2009 Page 14 of 34

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved.

#### 8 Component Type of a Java Implementation 611 An SCA runtime MUST introspect the componentType of a Java implementation class following the rules 612 defined in the section "Component Type of a Java Implementation". [JCI80001] 613 614 The component type of a Java Implementation is introspected from the implementation class as follows: 615 616 A <service/> element exists for each interface or implementation class identified by a @Service 617 annotation: 618 name attribute is the simple name of the interface or implementation class (i.e., without the 619 package name) 620 requires attribute is omitted unless the service implementation class is annotated with general or 621 specific intent annotations - in this case, the requires attribute is present with a value equivalent 622 to the intents declared by the service implementation class. 623 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 624 sets declared by the @PolicySets annotation. 625 626 <interface.java> child element is present with the interface attribute set to the fully gualified name 627 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 628 on Java interfaces, Java classes, and methods of Java interfaces are handled. 629 630 binding child element is omitted . 631 callback child element is omitted 632 633 A <reference/> element exists for each @Reference annotation: 634 name attribute has the value of the name parameter of the @Reference annotation, if present, . 635 otherwise it is the name of the field or the JavaBeans property name [JAVABEANS] 636 corresponding to the setter method name, depending on what element of the class is annotated 637 by the @Reference (note: for a constructor parameter, the @Reference annotation needs to have 638 a name parameter) 639 autowire attribute is omitted wiredByImpl attribute is omitted 640 641 target attribute is omitted 642 a) where the type of the field, setter or constructor parameter is an interface, the multiplicity 643 attribute is (1..1) unless the @Reference annotation contains required=false, in which case it 644 is (0..1) 645 b) where the type of the field, setter or parameter is an array or is a java.util.Collection, the 646 multiplicity attribute is (1..n) unless the @Reference annotation contains required=false, in 647 which case it is (0..n) 648 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 649 650 equivalent to the intents declared by the Java reference. 651 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 652 653 sets declared by the @PolicySets annotation. <interface.java> child element with the interface attribute set to the fully qualified name of the 654 interface class which types the field or setter method. See the Java Common Annotations and 655

30th April 2009 Page 15 of 34 Deleted: 08 Deleted: 27

656 657	APIs specification [JAVACAA] for a definition of how policy annotations on Java interfaces and methods of Java interfaces are handled.	
658	binding child element is omitted	
659	callback child element is omitted	
660		
661	A <property></property> element exists for each @Property annotation:	
662 663 664 665 666	<ul> <li>name attribute has the value of the name parameter of the @Property 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 @Property (note: for a constructor parameter, the @Property annotation needs to have a name parameter)</li> </ul>	
667	value attribute is omitted	
668 669 670 671   672 673	<ul> <li>type attribute which is set to the XML type implied by the JAXB mapping of the Java type of the field or the Java type defined by the parameter of the setter method. Where the type of the field or of the setter method is an array, the element type of the array is used. Where the type of the field or of the setter method is a java util.Collection, the parameterized type of the Collection or its member type is used. If the JAXB mapping is to a global element rather than a type (JAXB @XMLRootElement annotation), the type attribute is omitted.</li> </ul>	Deleted: n
674 675 676 677	<ul> <li>element attribute is omitted unless the JAXB mapping of the Java type of the field or the Java type defined by the parameter of the setter method is to a global element (JAXB @XMLRootElement annotation). In this case, the element attribute has the value of the name of the XSD global element implied by the JAXB mapping.</li> </ul>	
678 679	<ul> <li>many attribute is set to "false" unless the type of the field or of the setter method is an array or a java.util.Collection, in which case it is set to "true".</li> </ul>	
680 681	<ul> <li>mustSupply attribute is set to "true" unless the @Property annotation has required=false, in which case it is set to "false"</li> </ul>	
682		
683 684	An <implementation.java></implementation.java> element exists if the service implementation class is annotated with general or specific intent annotations or with @PolicySets:	
685 686 687	<ul> <li>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.</li> </ul>	
688 689 690	<ul> <li>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.</li> </ul>	
691	8.1 Component Type of an Implementation with no @Service	
692	Annotations	Deleted: a
693 694 695	The section defines the rules for determining the services of a Java component implementation that does not explicitly declare them using the @Service annotation. Note that these rules apply only to implementation classes that contain <b>no</b> @Service annotations.	
696 697	If there are no SCA services specified with the @Service annotation in an implementation class, the class offers:	
698 699	<ul> <li>either: one Service for each of the interfaces implemented by the class where the interface is annotated with @Remotable.</li> </ul>	Formatted: Indent: Left: 0.63 cm, Hanging: 0.63 cm, Tab stops: Not at
700 701 702	<ul> <li>or: if the class implements zero interfaces where the interface is annotated with @Remotable, then by default the implementation offers a single local service whose type is the implementation class itself</li> </ul>	1.9 cm
703	A <service></service> element exists for each service identified in this way:	Deleted: 27
	sca-javaci-1.1-spec-wd <u>10,</u> Copyright © OASIS® 2005,2009. All Rights Reserved. Page 16 of 34	

707 708 709	<ul> <li>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.</li> </ul>	
710 711 712	<ul> <li>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.</li> </ul>	
713 714 715 716	<ul> <li><interface.java> child element is present with the interface attribute set to the fully qualified name of the interface class or to the fully qualified name of the class itself. 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.</interface.java></li> </ul>	
717	binding child element is omitted	
718	callback child element is omitted	
719	8.2 ComponentType of an Implementation with no @Reference or	
720	@Property Annotations	Deleted: a
721 722 723 724	The section defines the rules for determining the properties and the references of a Java component implementation that does not explicitly declare them using the @Reference or the @Property annotations. Note that these rules apply only to implementation classes that contain <b>no</b> @Reference annotations <b>and no</b> @Property annotations.	
725		
726 727	In the absence of any @Property or @Reference annotations, the properties and references of an implementation class are defined as follows:	
728	The following setter methods and fields are taken into consideration:	
729 730	<ol> <li>Public setter methods that are not part of the implementation of an SCA service (either explicitly marked with @Service or implicitly defined as described above)</li> </ol>	Formatted: Indent: Left: 0.63 cm, Hanging: 0.63 cm
731	2. Public or protected fields unless there is a public setter method for the same name	
732		
733	An unannotated field or setter method is a <i>reference</i> if:	
734	its type is an interface annotated with @Remotable	Formatted: Indent: Left: 0.63 cm,
735 736	<ul> <li>its type is an array where the element type of the array is an interface annotated with @Remotable</li> </ul>	Hanging: 0.63 cm, Tab stops: Not at 1.9 cm + 13.76 cm
737 738	<ul> <li>its type is a java.util.Collection where the parameterized type of the Collection or its member type is an interface annotated with @Remotable</li> </ul>	
739	The reference in the component type has:	
740 741	<ul> <li>name attribute with the value of the name of the field or the JavaBeans property name [JAVABEANS] corresponding to the setter method name</li> </ul>	Formatted: Indent: Left: 0.63 cm, Hanging: 0.63 cm, Tab stops: Not at
742 743 744	<ul> <li>multiplicity attribute is (11) for the case where the type is an interface multiplicity attribute is (1n) for the cases where the type is an array or is a java.util.Collection</li> </ul>	1.9 cm + 13.76 cm
745 746 747 748	<ul> <li><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 APIs specification [JAVACAA] for a definition of how policy annotations on Java interfaces and methods of Java interfaces are handled.</interface.java></li> </ul>	
749 750	<ul> <li>requires attribute is omitted unless the field or setter method is also annotated with general or specific intent annotations - in this case, the requires attribute is present with a value</li> </ul>	
751	equivalent to the intents declared by the Java reference.	Deleted: 08
ĺ	sca-javaci-1.1-spec-wd10, 30th April 2009	Deleted: 27
ļ	Copyright © OASIS® 2005,2009. All Rights Reserved. Page 17 of 34	

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

753 754 755	<ul> <li>policySets attribute is omitted unless the field or setter method 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.</li> </ul>	
756	all other attributes and child elements of the reference are omitted	
757		
758	An unannotated field or setter method is a <i>property</i> if it is not a reference following the rules above.	
759	For each property of this type, the component type has a property element with:	
760	<ul> <li>name attribute with the value of the name of the field or the JavaBeans property name</li> </ul>	Formatted: Indent: Left: 0.63 cm,
761	[JAVABEANS] corresponding to the setter method name	Hanging: 0.63 cm, Tab stops: Not at 1.9 cm + 13.76 cm
762 763	<ul> <li>type attribute and element attribute set as described for a property declared via a @Property annotation</li> </ul>	
764	value attribute omitted	
765 766	<ul> <li>many attribute set to "false" unless the type of the field or of the setter method is an array or a java.util.Collection, in which case it is set to "true".</li> </ul>	
767	mustSupply attribute set to true	
768	8.3 Component Type Introspection Examples	
769 770	Example 8.1 shows how intent annotations can be applied to service and reference interfaces and methods as well as to a service implementation class.	
771	// Service interface	
772	package test;	
773	import org.oasisopen.sca.annotation.Authentication;	
774 775	<pre>import org.oasisopen.sca.annotation.Confidentiality;</pre>	
776	@Authentication	
777	public interface MyService {	
778	@Confidentiality	
779	void mymethod();	
780	}	
781		
782	// Reference interface	
783	package test;	
784	<pre>import org.oasisopen.sca.annotation.Integrity;</pre>	
785 786		
787	<pre>public interface MyRefInt {     @Integrity</pre>	
788	<pre>woid mymethod1();</pre>	
789	}	
790	ſ	
791	// Service implementation class	
792	package test;	
793	<pre>import static org.oasisopen.sca.Constants.SCA_PREFIX;</pre>	
794	<pre>import org.oasisopen.sca.annotation.Confidentiality;</pre>	
795	<pre>import org.oasisopen.sca.annotation.Reference;</pre>	
796	import org.oasisopen.sca.annotation.Service;	
797	<pre>@Service(MyService.class)</pre>	
798	<pre>@Requires(SCA_PREFIX+"managedTransaction")</pre>	
799	<pre>public class MyServiceImpl {</pre>	
800 801	<pre>@Confidentiality @Reference</pre>	
802	<pre>@Reference protected MyRefInt myRef;</pre>	
803	proceetea mynerine myner/	Deleted: 08
804	<pre>public void mymethod() {}</pre>	Deleted: 27
ĺ		Deleteu. 21
l	sca-javaci-1.1-spec-wd10, 30th April 2009 Copyright © OASIS® 2005,2009. All Rights Reserved. Page 18 of 34	

```
805
          }
806
      Example 8.1. Intent annotations on Java interfaces, methods, and implementations.
807
      Example 8.2 shows the introspected component type that is produced by applying the component type
808
      introspection rules to the interfaces and implementation from example 8.1.
809
          <componentType xmlns:sca=
810
                     "http://docs.oasis-open.org/ns/opencsa/sca/200903">
811
              <implementation.java class="test.MyServiceImpl"
812
                       requires="sca:managedTransaction"/>
813
              <service name="MyService" requires="sca:managedTransaction">
814
                   <interface.java interface="test.MyService"/>
815
              </service>
816
              <reference name="myRef" requires="sca:confidentiality">
817
                   <interface.java interface="test.MyRefInt"/>
818
              </reference>
819
          </componentType>
                                                                                                      Formatted: Indent: Left: 0.63 cm
820
      Example 8.2. Introspected component type with intents.
                                                                                                      Deleted: c
      8.4 Java Implementation with Conflicting Setter Methods
821
                                                                                                      Deleted: s
822
      If a Java implementation class, with or without @Property and @Reference annotations, has more than
                                                                                                      Deleted: m
      one setter method with the same JavaBeans property name [JAVABEANS] corresponding to the setter
823
      method name, then if more than one method is inferred to set the same SCA property or to set the same
824
      SCA reference, the SCA runtime MUST raise an error and MUST NOT instantiate the implementation
825
      class. [JCI80002]
826
827
      The following are examples of illegal Java implementation due to the presence of more than one setter
828
      method resulting in either an SCA property or an SCA reference with the same name:
829
830
          /** Illegal since two setter methods with same JavaBeans property name
                                                                                                     Formatted: Indent: Left: 0.63 cm
831
              are annotated with @Property annotation. */
                                                                                                      Deleted:
832
          public class IllegalImpl1 {
833
              // Setter method with upper case initial letter 'S'
834
              @Property
835
              public void setSomeProperty(String someProperty) {...}
836
              // Setter method with lower case initial letter 's'
837
838
              @Property
839
              public void setsomeProperty(String someProperty) {...}
840
          }
841
842
          /** Illegal since setter methods with same JavaBeans property name
           * are annotated with @Reference annotation. */
843
844
          public class IllegalImpl2 {
845
              // Setter method with upper case initial letter 'S'
846
              @Reference
847
              public void setSomeReference(SomeService service) {...}
848
849
              // Setter method with lower case initial letter 's'
850
              @Reference
851
              public void setsomeReference(SomeService service) {...}
852
          }
853
          /** Illegal since two setter methods with same JavaBeans property name
854
855
           * are resulting in an SCA property. Implementation has no @Property
          * or @Reference annotations. */
856
                                                                                                      Deleted: 08
857
          public class IllegalImpl3 {
                                                                                                      Deleted: 27
      sca-javaci-1.1-spec-wd10
                                                                                     30th April 2009
      Copyright © OASIS® 2005,2009. All Rights Reserved.
                                                                                      Page 19 of 34
```

```
862
              // Setter method with upper case initial letter 'S'
863
              public void setSomeOtherProperty(String someProperty) {...}
864
865
              // Setter method with lower case initial letter 's'
866
              public void setsomeOtherProperty(String someProperty) {...}
867
         }
868
869
         /\ensuremath{^{\ast\ast}} Illegal since two setter methods with same JavaBeans property name
         * are resulting in an SCA reference. Implementation has no @Property
* or @Reference annotations. */
870
871
872
         public class IllegalImpl4 {
873
              // Setter method with upper case initial letter 'S'
874
              public void setSomeOtherReference(SomeService service) {...}
875
876
              // Setter method with lower case initial letter 's'
877
              public void setsomeOtherReference(SomeService service) {...}
         }
878
879
880
      The following is an example of a legal Java implementation in spite of the implementation class having
881
      two setter methods with same JavaBeans property name [JAVABEANS] corresponding to the setter
882
      method name:
883
884
         /** Two setter methods with same JavaBeans property name, but one is
                                                                                                    Formatted: Indent: Left: 0.63 cm
         * annotated with @Property and the other is annotated with @Reference
* annotation. */
885
886
             annotation. */
887
         public class WeirdButLegalImpl {
888
              // Setter method with upper case initial letter 'F'
889
              @Property
890
              public void setFoo(String foo) {...}
891
892
              // Setter method with lower case initial letter 'f'
893
              @Reference
894
              public void setfoo(SomeService service) {...}
895
         }
896
```

Deleted: 08

Deleted: 27

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved.

897 898	9 Specifying the Java Implementation Type in an Assembly	_	
899 900	The following pseudo-schema defines the implementation element schema used for the Java implementation type:.	۹	Formatted: Indent: Left: 0 cm
901			
902 903	<implementation.java <br="" class="xs:NCName">requires="list of xs:QName"?</implementation.java>	<b></b> -	Formatted: Indent: Left: 0.63 cm
904 905	<pre>policySets="list of xs:QName"?/&gt;</pre>	< ◆	Formatted: Indent: Left: 4.44 cm, First line: 0.64 cm
906	The implementation.java element has the following attributes:		Formatted: Indent: Left: 0 cm
907	• class : NCName (11) – the fully qualified name of the Java class of the implementation	<b></b>	Formatted: Indent: Left: 0.63 cm,
908 909	• <b>requires : OName (0n)</b> – a list of policy intents. See the Policy Framework specification [POLICY] for a description of this attribute.		Hanging: 0.63 cm, Tab stops: 0.63 cm, List tab + Not at 1.27 cm
910 911	• <b>policySets</b> : <b>QName</b> (0n) – a list of policy sets. See the Policy Framework specification [POLICY] for a description of this attribute.		
912		<b>4</b>	Formatted: Indent: Left: 0 cm
913 914 915	The <implementation.java> element MUST conform to the schema defined in sca-implementation- java.xsd. [JCI90001]</implementation.java>		
916 917 918	The fully qualified name of the Java class referenced by the @class attribute of <implementation.java></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]		
919 920	The Java class referenced by the @class attribute of <implementation.java></implementation.java> MUST conform to Java SE version 5.0. [JCI90003]		

sca-javaci-1.1-spec-wd10,	
Copyright © OASIS® 2005,2009. All Righ	ts Reserved.

921	10 Java Packaging and Deployment Model			
922 923 924	The SCA Assembly <u>Model</u> Specification [ASSEMBLY] describes the basic packaging model for SCA contributions in the chapter on Packaging and Deployment. This specification defines extensions to the basic model for SCA contributions that contain Java component implementations.			
925 926 927 928 929	The model for the import and export of Java classes follows the model for import-package and export- package defined by the OSGi Service Platform Core Specification [OSGi Core]. Similar to an OSGI bundle, an SCA contribution that contains Java classes represents a class loader boundary at runtime. That is, classes are loaded by a contribution specific class loader such that all contributions with visibility to those classes are using the same Class Objects in the JVM.			
930	10.1 Contribution Metadata Extensions			
931 932 933 934 935 936 937	SCA contributions can be self contained such that all the code and metadata needed to execute the components defined by the contribution is contained within the contribution. However, in larger projects, there is often a need to share artifacts across contributions. This is accomplished through the use of the import and export extension points as defined in the sca-contribution.xml document. An SCA contribution that needs to use a Java class from another contribution can declare the dependency via an <import.java></import.java> extension element, contained within a <contribution></contribution> element, as defined below:			
938	<pre><import.java ?="" location="xs:anyURI" package="xs:string"></import.java></pre>	(	Formatted: Indent: Left: 0.63	cm
939 940	The import.java element has the following attributes:			
941 942 943 945 945 946 947 948 949 950 950	<ul> <li><i>package : string (11)</i> – The name of one or more Java package(s) to use from another contribution. Where there is more than one package, the package names are separated by a comma ",".</li> <li>The package can have a <i>version number range</i> appended to it, separated from the package name by a semicolon ";" followed by the text "version=" and the version number range, for example: package="com.acme.package1;version=1.4.1" package="com.acme.package2;version=[1.2,1.3]"</li> <li>Version number range follows the format defined in the OSGi Core specification [OSGi Core]:</li> </ul>			
952 953 954 955 956 957 958 959 960 961 962	<ul> <li>[1.2,1.3] - enclosing square brackets - inclusive range meaning any version in the range from the lowest to the highest, including the lowest and the highest</li> <li>(1.3.1,2.4.1) - enclosing round brackets - exclusive range meaning any version in the range from the lowest to the highest but not including the lowest or the highest.</li> <li>1.4.1 - no enclosing brackets - implies any version at or later than the specified version number is acceptable - equivalent to [1.4.1, infinity)</li> <li>If no version is specified for an imported package, then it is assumed to have a version range of [0.0.0, infinity) - ie any version is acceptable.</li> </ul>			
963 964	<ul> <li>Iocation : anyURI (01) – The URI of the SCA contribution which is used to resolve the java packages for this import.</li> </ul>			
965 966 967 968	Each Java package that is imported into the contribution MUST be included in one and only one import.java element. [JCI100001] Multiple packages can be imported, either through specifying multiple packages in the @package attribute or through the presence of multiple import.java elements.			
969	The SCA runtime MUST ensure that the package used to satisfy an import matches the package name,	ſ		
970 971	the version number or version number range and (if present) the location specified on the import.java element [JCI100002]	1	Deleted: 08	
	sca-javaci-1.1-spec-wd10, 30th April 2009 /	11-1	Deleted: 27	
	Copyright © OASIS® 2005,2009. All Rights Reserved. Page 22 of 34			

972	An SCA contribution that wants to allow a Java package to be used by another contribution can		
973	declare the exposure via an <export.java></export.java> extension element as defined below:		
974 975	<export.java package="xs:string"></export.java>		Formatted: Indent: Left: 0.63 cm
976	The export.java element has the following attributes:		
977 978 979 980 981 982 983	<ul> <li>package : string (11) - The name of one or more Java package(s) to expose for sharing by another contribution. Where there is more than one package, the package names are separated by a comma ",".</li> <li>The package can have a version number appended to it, separated from the package name by a semicolon ";" followed by the text "version=" and the version number: package="com.acme.package1; version=1.4.1"</li> </ul>	1	Deleted: 0
984 985 986 987	The package can have a <b>uses directive</b> appended to it, separated from the package name by a semicolon ";" followed by the text "uses=" which is then followed by a list of package names contained within single quotes """ (needed as the list contains commas).		
988 989 990 991 992 993 994 995 996	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. [JCI100003] Typically, the packages in the uses directive are packages used in the interface to the package being exported (eg as parameters or as classes/interfaces that are extended by the exported package). Example: package="com.acme.package1; uses='com.acme.package2,com.acme.package3'"		
990 997	If no version information is specified for an exported package, the version defaults to 0.0.0.		
998 999	If no uses directive is specified for an exported package, there is no requirement placed on a contribution which imports the package to use any particular version of any other packages.		
1000	Each Java package that is exported from the contribution MUST be included in one and only one		
1001 1002 1003	export.java element. [JCI100004] Multiple packages can be exported, either through specifying multiple packages in the @package attribute or through the presence of multiple export.java elements.		
1004	For example, a contribution that wants to:		
1005	• use classes from the <i>some.package</i> package from another contribution (any version)		
1006 1007	<ul> <li>use classes of the <i>some.other.package</i> package from another contribution, at exactly version 2.0.0</li> </ul>		
1008	• expose the my.package package from its own contribution, with version set to 1.0.0		
1009	would specify an sca-contribution.xml file as follows:		
1010			
1011 1012	<pre><?xml version="1.0" encoding="UTF-8"?> <contribution xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"></contribution></pre>		Formatted: Indent: Left: 0.63 cm
1013 1014	<import.java ,="" package="some.package"></import.java>		Deleted: "
1015	<pre><import.java package="some.other.package;version=[2.0.0]_"></import.java></pre>		Deleted: "
1016	<pre><export.java package="my.package;version=1.0.0"></export.java> </pre>	1.1	
1017			Deleted: "
1018	A love peakage that is appointed on an expert element MUCT be contained within the contribution		Deleted: "
1019 1020	A Java package that is specified on an export element MUST be contained within the contribution containing the export element. [JCI100007]		Deleted: "
1021			Deleted: "
		1	Deleted: 08
		11	Deleted: 27
	sca-javaci-1.1-spec-wd10, 30th April 2009 Copyright © OASIS® 2005,2009. All Rights Reserved. Page 23 of 34		

### 1029 10.2 Java Artifact Resolution

1030 1031	The SCA runtime MUST ensure that within a contribution, Java classes are resolved according to the following steps in the order specified:	,
1032 1033 1034	<ol> <li>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> </ol>	*'
1035	<ol><li>If the package of the Java class is specified in an import declaration then:</li></ol>	/
1036 1037	<ul> <li>a) if @location is specified, the location searched for the class is the contribution declared by the @location attribute.</li> </ul>	•
1038 1039 1040 1041	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.	
1042	If the Java package is not found, continue to step 3.	4
1043 1044	<ol> <li>The contribution itself is searched using the archive resolution rules defined by the Java Language.</li> </ol>	
1045	[JCI100008]	
1046	10.3 Class Loader Model	- 1 - 1
1047 1048 1049 1050	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. [JCI100010] The SCA runtime MUST ensure that Java classes that are imported into a contribution are loaded by the exporting contribution's class loader [JCI100011], as described in the section "Contribution Metadata Extensions"	
1051 1052	For example, suppose contribution A using class loader ACL, imports package some.package from contribution B that is using class loader BCL then the expression:	
1053	ACL.loadClass(importedClassName) == BCL.loadClass(importedClassName)	
1054	evaluates to true.	
1055 1056	The SCA runtime MUST set the thread context class loader of a component implementation class to the class loader of its containing contribution. [JCI100009]	1 4 1

1057

30th April 2009

Page 24 of 34

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved. Formatted: Indent: Left: 1.27 cm, Hanging: 0.63 cm

Hanging: 0.64 cm

Formatted: Indent: Left: 0.63 cm,

Formatted: Indent: Left: 0.76 cm

Deleted: The SCA runtime MUST ensure that within a contribution, Java classes are resolved according to the following steps in the order specified:¶ 1. If the contribution contains a Java Language specific resolution mechanism such as a classpath declaration in the archive's manifest, then that mechanism is used first to resolve classes. If the class is not found, then continue searching at step 2.5

2. If the package of the Java class is specified in an import declaration then: a) if @ location is specified, the location searched for the class is the contribution declared by the @location attribute.

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

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

Deleted: The SCA runtime MUST ensure that within a contribution, Java classes are resolved according to the following steps in the order specified:¶ 1. If the contribution contains a Java Language specific resolution mechanism such as a classpath declaration in the archive's manifest, then that mechanism is used first to resolve classes. If the class is not found, then continue searching at step 2.¶

 If the package of the Java class is specified in an import declaration then:¶
 a) if @location is specified, the location searched for the class is the contribution declared by the @location

attribute. ¶ b) if @location is not specified, the locations which are searched for the class are the contribution(s) in t(...[11])

Deleted: I

Deleted: ;

Formatted: Indent: Left: 0.63 cm, First line: 0 cm

Deleted: 08

# 1147 **11 Conformance**

1148 The XML schema pointed to by the RDDL document at the namespace URI, defined by this

specification, are considered to be authoritative and take precedence over the XML schema defined inthe appendix of this document.

1151

1152 There are three categories of artifacts that this specification defines conformance for: SCA Java

1153 Component Implementation Composite Document, SCA Java Component Implementation Contribution
 1154 Document and SCA Runtime.

### 1155 **11.1 SCA Java Component Implementation Composite Document**

- 1156 An SCA Java Component Implementation Composite Document is an SCA Composite Document, as
- 1157 defined by the SCA Assembly Model Specification Section 13.1 [ASSEMBLY], that uses the
- 1158 <implementation.java> element. Such an SCA Java Component Implementation Composite Document
- 1159 MUST be a conformant SCA Composite Document, as defined by [ASSEMBLY], and MUST comply with
- 1160 the requirements specified in Section 9 of this specification.

### 1161 **11.2 SCA Java Component Implementation Contribution Document**

1162 An SCA Java Component Implementation Contribution Document is an SCA Contribution Document, as

- 1163 defined by the SCA Assembly <u>Model</u> specification Section 13.1 [ASSEMBLY], that uses the contribution
- 1164 metadata extensions defined in Section 10. Such an SCA Java Component Implementation
- 1165 Contribution document MUST be a conformant SCA Contribution Document, as defined by
- 1166 [ASSEMBLY], and MUST comply with the requirements specified in Section 10 of this specification.

### 1167 **11.3 SCA Runtime**

- 1168 An implementation that claims to conform to this specification MUST meet the following conditions: 1169
- 1170
   1. The implementation MUST meet all the conformance requirements defined by the SCA

   1171
   Assembly Model Specification [ASSEMBLY].
- 11722. The implementation MUST reject an SCA Java Composite Document that does not conform to<br/>the sca-implementation-java.xsd schema.
- 11743. The implementation MUST reject an SCA Java Contribution Document that does not conform to<br/>the sca-contribution-java.xsd schema.
- 1176
   4. The implementation MUST meet all the conformance requirements, specified in 'Section 11\_\_\_\_

   1177
   Conformance', from the SCA Java Common Annotations and APIs Specification [JAVACAA].
- 1178 5. This specification permits an implementation class to use any and all the APIs and annotations defined in the Java Common Annotations and APIs Specification [JAVACAA], therefore the implementation MUST comply with all the statements in Appendix B: Conformance Items of [JAVACAA], notably all mandatory statements have to be implemented.
- 11826. The implementation MUST comply with all statements related to an SCA Runtime, specified in<br/>'Appendix B; Conformance Items' of this specification, notably all mandatory statements have<br/>to be implemented.1184to be implemented.
- 1185

6.46 cm + 8.08 cm + 9.69 cm + 11.31 cm + 12.92 cm + 14.54 cm + 16.16 cm + 17.77 cm + 19.39 cm + 21 cm + 22.62 cm + 24.23 cm + 25.85 cm Deleted: 2. Deleted: 3. Deleted: 4. Deleted: 5. Deleted: 5. Deleted: 5. Deleted: 5.

Formatted: Indent: Left: 0.63 cm,

Style: 1, 2, 3, ... + Start at: 1 +

Hanging: 0.63 cm, Space After: 6 pt, Numbered + Level: 1 + Numbering

Alignment: Left + Aligned at: 1.4 cm + Indent at: 2.03 cm, Tab stops: Not

at 1.62 cm + 3.23 cm + 4.85 cm +

,1	Deleted: 08
1	Deleted: 27

sca-javaci-1.1-spec-wd<u>10</u> Copyright © OASIS® 2005,2009. All Rights Reserved. 30th April 2009 Page 25 of 34 Deleted: s

# 1194 A. XML Schemas

### 1195 A.1 sca-contribution-java.xsd

```
1196
          <?xml version="1.0" encoding="UTF-8"?>
                                                                                              Formatted: Indent: Left: 0.63 cm
1197
          <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
1198
               OASIS trademark, IPR and other policies apply.
1199
          <schema xmlns="http://www.w3.org/2001/XMLSchema"</pre>
1200
             xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1201
             targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1202
             elementFormDefault="qualified">
1203
1204
             <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
1205
1206
             <!-- Import.java -->
1207
             <element name="import.java" type="sca:JavaImportType"/>
1208
             <complexType name="JavaImportType">
1209
                <complexContent>
1210
                   <extension base="sca:Import">
                      <attribute name="package" type="NCName" use="required"/>
1211
1212
                       <attribute name="location" type="anyURI" use="optional"/>
1213
                   </extension>
1214
                </complexContent>
1215
             </complexType>
1216
1217
             <!-- Export.java -->
1218
             <element name="export.java" type="sca:JavaExportType"/>
1219
             <complexType name="JavaExportType">
1220
                <complexContent>
1221
                   <extension base="sca:Export">
1222
                      <attribute name="package" type="NCName" use="required"/>
1223
                   </extension>
1224
                </complexContent>
1225
             </complexType>
1226
1227
          </schema>
1228
       A.2 sca-implementation-java.xsd
1229
          <?xml version="1.0" encoding="UTF-8"?>
                                                                                               Formatted: Indent: Left: 0.63 cm
1230
          <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
1231
               OASIS trademark, IPR and other policies apply. -->
1232
          <schema xmlns="http://www.w3.org/2001/XMLSchema"
             xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1233
1234
             targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1235
             elementFormDefault="qualified">
1236
1237
             <include schemaLocation="sca-core-1.1-cd03.xsd"/>
1238
1239
             <!-- Java Implementation -->
1240
             <element name="implementation.java" type="sca:JavaImplementation"</pre>
1241
                      substitutionGroup="sca:implementation"/>
1242
             <complexType name="JavaImplementation">
1243
                <complexContent>
1244
                   <extension base="sca:Implementation">
                                                                                               Deleted: 08
1245
                                                                                               Deleted: 27
```

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved. <u>30th April 2009</u> Page 26 of 34

1246 1247 1248 1249 1250 1251 1252 1253 1254 1255	<pre><sequence></sequence></pre>	Deleted: ¶
1255		

Deleted: 08

Deleted: 27

sca-javaci-1.1-spec-wd<u>10</u> Copyright © OASIS® 2005,2009. All Rights Reserved. 30th April 2009 Page 27 of 34

# 1259 **B. Conformance Items**

1260 This section contains a list of conformance items for the SCA Java Component Implementation1261 specification.

### 1262

Conformance ID	Description		Formatted Table
[JCI20001]	The services provided by a Java-based implementation MUST have an interface defined in one of the following ways:		Deleted: [JCl20001] [12]
	A Java interface		Formatted: Bulleted + Level: 1 +
	• <u>A Java class</u>		Aligned at: 0.63 cm + Indent at: 1.27 cm
	<ul> <li>A Java interface generated from a Web Services Description Language [WSDL] (WSDL) portType.</li> </ul>		Formatted: English (United States)
[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.	1	
[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		Deleted: [JCI50003]
	one of two ways;		Deleted: ¶
	<ul> <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> </ul>		Formatted: Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm
	<ul> <li>The container can inject a proxy to the target service; invocation of methods on the proxy can result in a ServiceUnavailableException</li> </ul>		Deleted: ¶ •
[JCI50004]	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.		
	<ol> <li>A declared constructor, all of whose parameters are annotated with either @Property or @Reference.</li> </ol>		
¥	3. A no-argument constructor.		Formatted: Indent: Left: 0.63 cm,
[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.		Hanging: 0.64 cm Deleted: [JCl50004]
[JCI60001]	The SCA runtime MUST support the STATELESS and COMPOSITE implementation scopes.	1	
[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	,	Deleted: 08
		11	Deleted: 27

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved. 30th April 2009 Page 28 of 34

Ī		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.		
	[JCI90001]	The <implementation.java> element MUST conform to the schema defined in sca-implementation-java.xsd.</implementation.java>		
	[JCI90002]	The fully qualified name of the Java class referenced by the @class attribute of <implementation.java></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></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]	The SCA runtime MUST ensure that within a contribution, Java classes are resolved according to the following steps in the order specified:		
		1. If the contribution contains a Java Language specific resolution mechanism such as a classpath declaration in the archive's manifest, then that mechanism is used first to resolve classes. If the class is not	````	Formatted: Indent: Left: 0.63 cm, Hanging: 0.64 cm Deleted:
		found, then continue searching at step 2.		
		2. If the package of the Java class is specified in an import declaration then:		Deleted:
1		<ul> <li>a) if @location is specified, the location searched for the class is the contribution declared by the @location attribute.</li> </ul>		Formatted: Indent: Left: 1.27 cm, Hanging: 0.63 cm
I		b) if @location is not specified, the locations which are searched for the class are the contribution(s) in the Domain which have export		Deleted:
		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.		Deleted:
		If the Java package is not found, continue to step 3.		Deleted: j
		3. The contribution itself is searched using the archive resolution rules defined by the Java Language.	`\\	Formatted: Indent: Left: 0.63 cm, Hanging: 0.64 cm
	[JCI100009]	The SCA runtime MUST set the thread context class loader of a component implementation class to the class loader of its containing contribution.		Deleted:
	[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.		Deleted: 08
			1%	Deleted: 27
I	sca-javaci-1.1-spec-wd10,	<u>30</u> th April 2009	11	

sca-javaci-1.1-spec-wd<u>10</u> Copyright © OASIS® 2005,2009. All Rights Reserved.

30th April 2009 Page 29 of 34

) )

)

[JCI100011]

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

1276

Deleted: 08

Deleted: 27

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved.



### C. Acknowledgements 1277

1278 1279 The following individuals have participated in the creation of this specification and are gratefully

acknowledged:

#### 1280 Participants:

Participant Name	Affiliation
Bryan Aupperle	IBM
Ron Barack	SAP AG*
Michael Beisiegel	IBM
Henning Blohm	SAP AG*
David Booz	IBM
Martin Chapman	Oracle Corporation
Graham Charters	IBM
Shih-Chang Chen	Oracle Corporation
Chris Cheng	Primeton Technologies, Inc.
Vamsavardhana Reddy Chillakuru	IBM
Roberto Chinnici	Sun Microsystems
Pyounguk Cho	Oracle Corporation
Eric Clairambault	IBM
Mark Combellack	Avaya, Inc.
Jean-Sebastien Delfino	IBM
Mike Edwards	IBM
Raymond Feng	IBM
Bo Ji	Primeton Technologies, Inc.
Uday Joshi	Oracle Corporation
Anish Karmarkar	Oracle Corporation
Michael Keith	Oracle Corporation
Rainer Kerth	SAP AG*
Meeraj Kunnumpurath	Individual
Simon Laws	IBM
Yang Lei	IBM
Mark Little	Red Hat
Ashok Malhotra	Oracle Corporation
Jim Marino	Individual
Jeff Mischkinsky	Oracle Corporation
Sriram Narasimhan	TIBCO Software Inc.
Simon Nash	Individual
Sanjay Patil	SAP AG*
Plamen Pavlov	SAP AG*
Peter Peshev	SAP AG*
Ramkumar Ramalingam	IBM
Luciano Resende	IBM
Michael Rowley	Active Endpoints, Inc.
Vladimir Savchenko	SAP AG*
Pradeep Simha	TIBCO Software Inc.
Raghav Srinivasan	Oracle Corporation
sca-javaci-1 1-spec-wd10	

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved.

30th April 2009 Page 31 of 34

Deleted: 08

Scott Vorthmann Feng Wang Robin Yang

1281

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

Deleted: 08

Deleted: 27

30th April 2009 Page 32 of 34

sca-javaci-1.1-spec-wd<u>10</u> Copyright © OASIS® 2005,2009. All Rights Reserved.

# 1282 **D. Non-Normative Text**

|--|

Deleted: 27

sca-javaci-1.1-spec-wd<u>10,</u> Copyright © OASIS® 2005,2009. All Rights Reserved. 30th April 2009 Page 33 of 34

# 1283 E. Revision History

1284

[optional; should not be included in OASIS Standards]

1285

94	[optional, should not be included in OASIS Standards]	
35		

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	<ul> <li>* Applied resolution for issue 55, 32</li> <li>* Editorial cleanup to make a working draft</li> <li>- [1] style changed to [ASSEMBLY]</li> <li>- updated namespace references</li> </ul>
wd03	2009-02-26	David Booz	<ul><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
<u>wd09</u>	<u>2009-04-29</u>	David Booz	Editorial fixes throughout (capitalization, quotes, fonts, spec references, etc.)
<u>wd10</u>	<u>2009-04-30</u>	David Booz	Editorial fixes, indention, etc.

1286

1287

Deleted: 08

Deleted: 27

sca-javaci-1.1-spec-wd10, Copyright © OASIS® 2005,2009. All Rights Reserved.

Page 1: [1] Deleted	Dave Booz	30/04/2009 10:33:00
08		
Page 1: [1] Deleted	Dave Booz	30/04/2009 10:33:00
08		
Page 11: [2] Deleted	Simon Nash	02/05/2009 17:43:00
The constructor to use for the cre runtime using the sequence:	eation of an implementation instance N	IUST be selected by the SCA
1. A declared constructor a	nnotated with a @Constructor annotat	ion.
2. A declared constructor, a @Reference.	all of whose parameters are annotated	with either @Property or
	or.The constructor to use for the creat the SCA runtime using the sequence:	ion of an implementation
1. A declared constructor a	nnotated with a @Constructor annotat	ion.
2. A declared constructor, a @Reference.	all of whose parameters are annotated	with either @Property or
3. A no-argument construct	or.	
Page 11: [3] Formatted	Simon Nash	02/05/2009 16:52:00
Bulleted + Level: 1 + Aligned at:	0.63 cm + Indent at: 1.27 cm	
Page 11: [4] Deleted	Simon Nash	02/05/2009 18:07:00
<ul> <li>The container can inject result in a ServiceUnavailableEx the SCA runtime in one of two was</li> <li>If any reference in the cy construction, followed by inject</li> </ul>	cle is optional, then the container can action of a reference to the target befor a proxy to the target service; invoca	n of methods on the proxy can ponents MUST be handled by inject a null value during re invoking any service.
Dage 11, [5] Formatted	Simon Noch	02/05/2000 21.52.00
Page 11: [5] Formatted Indent: Left: 0.63 cm, Space Afte	Simon Nash er: 0 pt	02/05/2009 21:53:00
Page 11: [6] Formatted	Simon Nash	02/05/2009 21:51:00
Font: (Default) Courier New, 10 p	ot	
Page 11: [7] Formatted	Simon Nash	02/05/2009 21:51:00
Font: 10 pt		
Page 11: [7] Formatted	Simon Nash	02/05/2009 21:51:00
Font: 10 pt		
Page 11: [7] Formatted	Simon Nash	02/05/2009 21:51:00
Font: 10 pt		
Page 11: [8] Deleted	Simon Nash	02/05/2009 21:51:00
		02/03/2009 21.31.00
		02/03/2007/21.31.00

Page 11: [9] Deleted	Simon Nash	02/05/2009 21:52:00
Page 11: [9] Deleted	Simon Nash	02/05/2009 21:52:00
Page 11: [10] Deleted	Dave Booz	30/04/2009 10:47:00

Page 24: [11] Deleted Simon Nash	02/05/2009 18:17:00
----------------------------------	---------------------

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

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

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

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

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

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

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

Page 28: [12] Deleted	Simon Nash	02/05/2009 17:20:00
[JCI20001]	The services provided by a Java-based i an interface defined in one of the followir	
	<ul> <li>A Java interface</li> </ul>	
	<ul> <li>A Java class</li> </ul>	
	• A Java interface generated from a Web S Language [WSDL] (WSDL) portType.	Services Description
Page 28: [13] Deleted	Simon Nash	02/05/2009 17:11:00
Page 28: [13] Deleted [JCI50004]	Simon Nash The constructor to use for the creation of MUST be selected by the SCA runtime u	f an implementation instance
	The constructor to use for the creation of	f an implementation instance using the sequence:
	The constructor to use for the creation of MUST be selected by the SCA runtime u	f an implementation instance using the sequence: th a @Constructor annotation.