



Service Component Architecture Java Common Annotations and APIs Specification Version 1.1

Committee Draft 02, Revision 03+Issue1 rev 8

29 June 2009

Deleted: 2

Deleted: 19

Deleted: 6

Deleted: March

Specification URIs:

This Version:

<http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd02-rev3.html>
<http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd02-rev3.doc>
<http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd02-rev3.pdf> (normative)

Previous Version:

Latest Version:

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

Latest Approved Version:

Technical Committee:

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

Chair(s):

Simon Nash,	IBM
Michael Rowley,	BEA Systems
Mark Combellack,	Avaya

Editor(s):

Ron Barack,	SAP
David Booz,	IBM
Mark Combellack,	Avaya
Mike Edwards,	IBM
Anish Karmarkar,	Oracle
Ashok Malhotra,	Oracle
Peter Peshev,	SAP

Related work:

This specification replaces or supersedes:

- Service Component Architecture Java Annotations and APIs Specification Version 1.00, March 21 2007

This specification is related to:

- Service Component Architecture Assembly Model Specification Version 1.1
- Service Component Architecture Policy Framework Specification Version 1.1

Deleted: March

Declared XML Namespace(s):

<http://docs.oasis-open.org/ns/opencsa/sca/200712>

Abstract:

The SCA Java Common Annotation and APIs specify a Java syntax for programming concepts defined in the SCA Assembly Model Specification. It specifies a set of APIs and annotations that can be used by Java-based SCA specifications.

Specifically, this specification covers:

1. Implementation metadata for specifying component services, references, and properties
2. A client and component API
3. Metadata for asynchronous services
4. Metadata for callbacks
5. Definitions of standard component implementation scopes
6. Java to WSDL and WSDL to Java mappings
7. Security policy annotations

Note that individual programming models can chose to implement their own mappings of assembly model concepts using native APIs and idioms when appropriate.

Status:

This document was last revised or approved by the OASIS Service Component Architecture / J (SCA-J) TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/sca-j/>.

For information on whether any patents have been disclosed that might be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/sca-j/ipr.php>).

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

Notices

Copyright © OASIS® 2005, 2008. All Rights Reserved.

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

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

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

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

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

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

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

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

Table of Contents

1	Introduction	7
1.1	Terminology	7
1.2	Normative References	7
1.3	Non-Normative References	8
2	Implementation Metadata	9
2.1	Service Metadata	9
2.1.1	@Service	9
2.1.2	Java Semantics of a Remotable Service	9
2.1.3	Java Semantics of a Local Service	9
2.1.4	@Reference	10
2.1.5	@Property	10
2.2	Implementation Scopes: @Scope, @Init, @Destroy	10
2.2.1	Stateless scope	10
2.2.2	Composite scope	11
3	Interface	12
3.1	Java interface element – <interface.java>	12
3.2	@Remotable	13
3.3	@Callback	13
4	Client API	14
4.1	Accessing Services from an SCA Component	14
4.1.1	Using the Component Context API	14
4.2	Accessing Services from non-SCA component implementations	14
4.2.1	SCAClient Interface and Related Classes	14
5	Error Handling	15
6	Asynchronous Programming	16
6.1	@OneWay	16
6.2	Callbacks	16
6.2.1	Using Callbacks	16
6.2.2	Callback Instance Management	18
6.2.3	Implementing Multiple Bidirectional Interfaces	18
6.2.4	Accessing Callbacks	19
7	Policy Annotations for Java	20
7.1	General Intent Annotations	20
7.2	Specific Intent Annotations	22
7.2.1	How to Create Specific Intent Annotations	22
7.3	Application of Intent Annotations	23
7.3.1	Inheritance And Annotation	23
7.4	Relationship of Declarative And Annotated Intents	25
7.5	Policy Set Annotations	25
7.6	Security Policy Annotations	26
7.6.1	Security Interaction Policy	26
7.6.2	Security Implementation Policy	27
8	Java API	30

8.1 Component Context.....	30	
8.2 Request Context.....	31	
8.3 ServiceReference.....	32	
8.4 ServiceRuntimeException.....	32	
8.5 ServiceUnavailableException.....	33	
8.6 InvalidServiceException.....	33	
8.7 Constants.....	33	
8.8 SCAClient Interface.....	33	
8.9 SCAClientFactory Class.....	34	Deleted: 36
8.10 NoSuchDomainException.....	36	Deleted: 37
8.11 NoSuchServiceException.....	38	Deleted: 37
9 Java Annotations.....	39	Deleted: 37
9.1 @AllowsPassByReference.....	39	Deleted: 37
9.2 @Authentication.....	39	Deleted: 38
9.3 @Callback.....	40	Deleted: 39
9.4 @ComponentName.....	41	Deleted: 40
9.5 @Confidentiality.....	42	Deleted: 41
9.6 @Constructor.....	43	Deleted: 41
9.7 @Context.....	43	Deleted: 42
9.8 @Destroy.....	44	Deleted: 42
9.9 @EagerInit.....	44	Deleted: 43
9.10 @Init.....	45	Deleted: 43
9.11 @Integrity.....	45	Deleted: 44
9.12 @Intent.....	46	Deleted: 45
9.13 @OneWay.....	47	Deleted: 45
9.14 @PolicySets.....	47	Deleted: 46
9.15 @Property.....	48	Deleted: 47
9.16 @Qualifier.....	49	Deleted: 48
9.17 @Reference.....	50	Deleted: 48
9.17.1 Reinjection.....	52	Deleted: 50
9.18 @Remotable.....	54	Deleted: 52
9.19 @Requires.....	55	Deleted: 53
9.20 @Scope.....	56	Deleted: 54
9.21 @Service.....	57	Deleted: 55
10 WSDL to Java and Java to WSDL.....	59	Deleted: 57
10.1 JAX-WS Client Asynchronous API for a Synchronous Service.....	59	Deleted: 57
A. XML Schema: sca-interface-java.xsd.....	61	Deleted: 59
B. Java Classes and Interfaces.....	62	Deleted: 60
B.1 SCAClient Classes and Interfaces.....	62	Deleted: 60
B.1.1 SCAClient Interface.....	62	Deleted: 60
B.1.2 SCAClientFactory Class.....	63	Deleted: 61
B.1.3 SCAFactoryFinder class.....	65	Deleted: 62
B.1.4 SCAClient Classes and Interfaces - what does a vendor need to do?.....	71	Deleted: 67
C. Conformance Items.....	72	Deleted: 69
D. Acknowledgements.....	78	Deleted: 75
		Deleted: March

E. Non-Normative Text 79
F. Revision History..... 80

Deleted: 76

Deleted: 77

Deleted: March

1 Introduction

The SCA Common Annotation, APIs, Client and Implementation Model specifies a Java syntax for programming concepts defined in the SCA Assembly Model Specification [ASSEMBLY]. It specifies a set of APIs and annotations that can be used by Java-based SCA specifications.

Specifically, this specification covers:

1. Implementation metadata for specifying component services, references, and properties
2. A client and component API
3. Metadata for asynchronous services
4. Metadata for callbacks
5. Definitions of standard component implementation scopes
6. Java to WSDL and WSDL to Java mappings
7. Security policy annotations

Note that individual programming models can choose to implement their own mappings of assembly model concepts using native APIs and idioms when appropriate.

The goal of specifying the annotations, APIs, client and implementation model in this specification is to promote consistency and reduce duplication across various Java-related component implementation type specifications. The annotations, APIs and client and implementation model defined in this specification are designed to be used by other SCA Java-related specifications in either a partial or complete fashion.

This document defines implementation metadata using the annotation capability from Java™ 2 Standard Edition (J2SE) 5. However, SCA also allows service clients and implementations to be written using J2SE 1.4. All metadata that is represented by annotations can also be expressed using a component type side file, as defined in the SCA Assembly Specification [ASSEMBLY].

Comment [ME1]: This sentence needs to be removed

1.1 Terminology

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

1.2 Normative References

- | | |
|------------|---|
| [RFC2119] | S. Bradner, <i>Key words for use in RFCs to Indicate Requirement Levels</i> , http://www.ietf.org/rfc/rfc2119.txt , IETF RFC 2119, March 1997. |
| [ASSEMBLY] | SCA Assembly Specification, http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd01.pdf |
| [SDO] | SDO 2.1 Specification, http://www.osoa.org/download/attachments/36/Java-SDO-Spec-v2.1.0-FINAL.pdf |
| [JAX-B] | JAXB 2.1 Specification, http://www.jcp.org/en/jsr/detail?id=222 |
| [WSDL] | WSDL Specification,
WSDL 1.1: http://www.w3.org/TR/wsdl ,
WSDL 2.0: http://www.w3.org/TR/wsdl20/ |
| [POLICY] | SCA Policy Framework, http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd-01.pdf |

Deleted: March

- 44 **[JSR-250]** Common Annotation for Java Platform specification (JSR-250),
45 <http://www.jcp.org/en/jsr/detail?id=250>
46 **[JAX-WS]** JAX-WS 2.1 Specification (JSR-224),
47 <http://www.jcp.org/en/jsr/detail?id=224>
48 **[JAVABEANS]** JavaBeans 1.01 Specification,
49 <http://java.sun.com/javase/technologies/desktop/javabeans/api/>
50

51 **1.3 Non-Normative References**

- 52 **[EBNF-Syntax]** Extended BNF syntax format used for formal grammar of constructs
53 <http://www.w3.org/TR/2004/REC-xml-20040204/#sec-notation>

Deleted: March

54 2 Implementation Metadata

55 This section describes SCA Java-based metadata, which applies to Java-based implementation
56 types.

57 2.1 Service Metadata

58 2.1.1 @Service

59 The **@Service annotation** is used on a Java class to specify the interfaces of the services
60 implemented by the implementation. Service interfaces are defined in one of the following ways:

- 61 • As a Java interface
- 62 • As a Java class
- 63 • As a Java interface generated from a Web Services Description Language [WSDL]
64 (WSDL) portType (Java interfaces generated from a WSDL portType are always
65 **remotable**)

66 2.1.2 Java Semantics of a Remotable Service

67 A **remotable service** is defined using the @Remotable annotation on the Java interface that
68 defines the service. Remotable services are intended to be used for **coarse grained** services, and
69 the parameters are passed **by-value**. **Remotable Services MUST NOT make use of *method***
70 ***overloading***. [JCA20001]

71 The following snippet shows an example of a Java interface for a remote service:

```
72 package services.hello;  
73 @Remotable  
74 public interface HelloService {  
75     String hello(String message);  
76 }
```

Deleted: Remotable Services MUST NOT make use of **method overloading**.

Deleted: Remotable Services MUST NOT make use of **method overloading**.

77 2.1.3 Java Semantics of a Local Service

78 A **local service** can only be called by clients that are deployed within the same address space as
79 the component implementing the local service.

80 A local interface is defined by a Java interface with no @Remotable annotation or it is defined by a
81 Java class.

82 The following snippet shows an example of a Java interface for a local service:

```
83 package services.hello;  
84 public interface HelloService {  
85     String hello(String message);  
86 }  
87
```

88 The style of local interfaces is typically **fine grained** and is intended for **tightly coupled**
89 interactions.

90 The data exchange semantic for calls to local services is **by-reference**. This means that
91 implementation code which uses a local interface needs to be written with the knowledge that
92 changes made to parameters (other than simple types) by either the client or the provider of the
93 service are visible to the other.

Deleted: March

94 **2.1.4 @Reference**

95 Accessing a service using reference injection is done by defining a field, a setter method
96 parameter, or a constructor parameter typed by the service interface and annotated with a
97 **@Reference** annotation.

98 **2.1.5 @Property**

99 Implementations can be configured with data values through the use of properties, as defined in
100 the SCA Assembly specification [ASSEMBLY]. The **@Property** annotation is used to define an SCA
101 property.

102 **2.2 Implementation Scopes: @Scope, @Init, @Destroy**

103 Component implementations can either manage their own state or allow the SCA runtime to do so.
104 In the latter case, SCA defines the concept of **implementation scope**, which specifies a visibility
105 and lifecycle contract an implementation has with the SCA runtime. Invocations on a service
106 offered by a component will be dispatched by the SCA runtime to an **implementation instance**
107 according to the semantics of its implementation scope.

108 Scopes are specified using the **@Scope** annotation on the implementation class.

109 This document defines two scopes:

- 110 • STATELESS
- 111 • COMPOSITE

112 Java-based implementation types can choose to support any of these scopes, and they can define
113 new scopes specific to their type.

114 An implementation type can allow component implementations to declare **lifecycle methods** that
115 are called when an implementation is instantiated or the scope is expired.

116 **@Init** denotes a method called upon first use of an instance during the lifetime of the scope
117 (except for composite scoped implementation marked to eagerly initialize, see [section Composite](#)
118 [Scope](#)).

119 **@Destroy** specifies a method called when the scope ends.

120 Note that only no argument methods with a void return type can be annotated as lifecycle
121 methods.

122 The following snippet is an example showing a fragment of a service implementation annotated
123 with lifecycle methods:

```
124 @Init
125 public void start() {
126     ...
127 }
128
129 @Destroy
130 public void stop() {
131     ...
132 }
133
134
```

135 The following sections specify the two standard scopes which a Java-based implementation type
136 can support.

137 **2.2.1 Stateless scope**

138 For stateless scope components, there is no implied correlation between implementation instances
139 used to dispatch service requests.

Deleted: March

140 The concurrency model for the stateless scope is single threaded. This means that the SCA
141 runtime MUST ensure that a stateless scoped implementation instance object is only ever
142 dispatched on one thread at any one time. [JCA20002] In addition, within the SCA lifecycle of a
143 stateless scoped implementation instance, the SCA runtime MUST only make a single invocation of
144 one business method. [JCA20003] Note that the SCA lifecycle might not correspond to the Java
145 object lifecycle due to runtime techniques such as pooling.

146 2.2.2 Composite scope

147 For a composite scope implementation instance, the SCA runtime MUST ensure that all service
148 requests are dispatched to the same implementation instance for the lifetime of the containing
149 composite. [JCA20004] The lifetime of the containing composite is defined as the time it becomes
150 active in the runtime to the time it is deactivated, either normally or abnormally.

151 ~~When the implementation class is marked for eager initialization, the SCA runtime MUST create a~~
152 ~~composite scoped instance when its containing component is started. [JCA20005] If a method of~~
153 ~~an implementation class is marked with the @Init annotation, the SCA runtime MUST call that~~
154 ~~method when the implementation instance is created. [JCA20006]~~

155 The concurrency model for the composite scope is multi-threaded. This means that the SCA
156 runtime MAY run multiple threads in a single composite scoped implementation instance object
157 and the SCA runtime MUST NOT perform any synchronization. [JCA20007]

Deleted: When the implementation class is marked for eager initialization, the SCA runtime MUST create a composite scoped instance when its containing component is started.

Deleted: When the implementation class is marked for eager initialization, the SCA runtime MUST create a composite scoped instance when its containing component is started.

Deleted: If a method of an implementation class is marked with the @Init annotation, the SCA runtime MUST call that method when the implementation instance is created.

Deleted: If a method of an implementation class is marked with the @Init annotation, the SCA runtime MUST call that method when the implementation instance is created.

Deleted: March

158 3 Interface

159 This section describes the SCA Java interface element and the SCA metadata for Java interfaces.

160 3.1 Java interface element – <interface.java>

161 The Java interface element is used in SCDL files in places where an interface is declared in terms
162 of a Java interface class. The Java interface element identifies the Java interface class and can
163 also identify a callback interface, where the first Java interface represents the forward (service)
164 call interface and the second interface represents the interface used to call back from the service
165 to the client.

166 The `interface.java` element MUST conform to the schema defined in the `sca-interface-java.xsd`
167 schema. [JCA30004]

168 The following is the pseudo-schema for the `interface.java` element

169

```
170 <interface.java interface="NCName" callbackInterface="NCName"? />
```

171

172 The `interface.java` element has the following attributes:

- 173 • **interface : NCName (1..1)** – the Java interface class to use for the service interface. The
174 value of the `@interface` attribute MUST be the fully qualified name of the Java interface
175 class. [JCA30001]
- 176 • **callbackInterface : NCName (0..1)** – the Java interface class to use for the callback
177 interface. The value of the `@callbackInterface` attribute MUST be the fully qualified name
178 of a Java interface used for callbacks. [JCA30002]

179

180 The following snippet shows an example of the Java interface element:

181

```
182 <interface.java interface="services.stockquote.StockQuoteService"  
183 callbackInterface="services.stockquote.StockQuoteServiceCallback" />
```

184

185 Here, the Java interface is defined in the Java class file

186 `./services/stockquote/StockQuoteService.class`, where the root directory is defined by the
187 contribution in which the interface exists. Similarly, the callback interface is defined in the Java
188 class file `./services/stockquote/StockQuoteServiceCallback.class`.

189 Note that the Java interface class identified by the `@interface` attribute can contain a Java
190 `@Callback` annotation which identifies a callback interface. If this is the case, then it is not
191 necessary to provide the `@callbackInterface` attribute. However, if the Java interface class
192 identified by the `@interface` attribute does contain a Java `@Callback` annotation, then the Java
193 interface class identified by the `@callbackInterface` attribute MUST be the same interface class.
194 [JCA30003]

195 For the Java interface type system, parameters and return types of the service methods are
196 described using Java classes or simple Java types. It is recommended that the Java Classes used
197 conform to the requirements of either JAXB [JAX-B] or of Service Data Objects [SDO] because of
198 their integration with XML technologies.

199

200

Deleted: The value of the
@interface attribute MUST
be the fully qualified name
of the Java interface class

Deleted: The value of the
@interface attribute MUST
be the fully qualified name
of the Java interface class

Deleted: The value of the
@callbackInterface
attribute MUST be the fully
qualified name of a Java
interface used for callbacks

Deleted: The value of the
@callbackInterface
attribute MUST be the fully
qualified name of a Java
interface used for callbacks

Deleted: March

201 3.2 @Remotable

202 The **@Remotable** annotation on a Java interface indicates that the interface is designed to be
203 used for remote communication. Remotable interfaces are intended to be used for **coarse**
204 **grained** services. Operations' parameters and return values are passed **by-value**. Remotable
205 Services are not allowed to make use of method **overloading**.

206 3.3 @Callback

207 A callback interface is declared by using a @Callback annotation on a Java service interface, with
208 the Java Class object of the callback interface as a parameter. There is another form of the
209 @Callback annotation, without any parameters, that specifies callback injection for a setter method
210 or a field of an implementation.

211 4 Client API

212 This section describes how SCA services can be programmatically accessed from components and
213 also from non-managed code, i.e. code not running as an SCA component.

214 4.1 Accessing Services from an SCA Component

215 An SCA component can obtain a service reference either through injection or programmatically
216 through the **ComponentContext** API. Using reference injection is the recommended way to
217 access a service, since it results in code with minimal use of middleware APIs. The
218 ComponentContext API is provided for use in cases where reference injection is not possible.

219 4.1.1 Using the Component Context API

220 When a component implementation needs access to a service where the reference to the service is
221 not known at compile time, the reference can be located using the component's
222 ComponentContext.

223 4.2 Accessing Services from non-SCA component implementations

224 This section describes how Java code not running as an SCA component that is part of an SCA
225 composite accesses SCA services via references.

226 4.2.1 SCAClient Interface and Related Classes

227 Client code can use the **SCAClient** interface to obtain proxy reference objects for a service which
228 is in an SCA Domain. The URI of the domain, the relative URI of the service and the business
229 interface of the service must all be known in order to use the SCAClient interface.

230 Objects which implement the SCAClient interface are obtained using the SCAClientFactory class.

232 The following is a sample of the code that a client would use:

```
233 import org.oasisopen.sca.client.SCAClient;  
234 import org.oasisopen.sca.client.SCAClientFactory;  
235 import com.foo.HelloService;  
236  
237 public void someMethod() {  
238  
239     String serviceURI = "SomeHelloServiceURI";  
240     URI domainURI = new URI("SomeDomainURI");  
241  
242     ...  
243     SCAClient scaClient = SCAClientFactory.newInstance();  
244     HelloService helloService =  
245         scaClient.getService(HelloService.class,  
246                             serviceURI, domainURI);  
247     String reply = helloService.sayHello("Mark");  
248     ...  
249 }
```

250 For details about the SCAClient interface and its related classes see the section "SCAClient
251 Interface" and the section "SCAClientFactory Class".

252

Deleted: ComponentContext

Deleted: Non-SCA client code can use the ComponentContext API to perform operations against a component in an SCA domain. How client code obtains a reference to a ComponentContext is runtime specific. ¶ The following example demonstrates the use of the component Context API by non-SCA code:

Deleted: d

Deleted: ¶

Deleted: ComponentContext context = // obtained via host environment-specific means ¶ HelloService helloService = ¶ context.getService(HelloService.class, "HelloService"); ¶ String result = helloService.hello("Hello World!"); ¶

Formatted: Bullets and Numbering

Deleted: March

253 5 Error Handling

254 Clients calling service methods can experience business exceptions and SCA runtime exceptions.

255 Business exceptions are thrown by the implementation of the called service method, and are
256 defined as checked exceptions on the interface that types the service.

257 SCA runtime exceptions are raised by the SCA runtime and signal problems in management of
258 component execution or problems interacting with remote services. The SCA runtime exceptions
259 are [defined in the Java API section](#).

260 6 Asynchronous Programming

261 Asynchronous programming of a service is where a client invokes a service and carries on
262 executing without waiting for the service to execute. Typically, the invoked service executes at
263 some later time. Output from the invoked service, if any, is fed back to the client through a
264 separate mechanism, since no output is available at the point where the service is invoked. This is
265 in contrast to the call-and-return style of synchronous programming, where the invoked service
266 executes and returns any output to the client before the client continues. The SCA asynchronous
267 programming model consists of:

- 268 • support for non-blocking method calls
- 269 • callbacks

270 Each of these topics is discussed in the following sections.

271 6.1 @OneWay

272 **Nonblocking calls** represent the simplest form of asynchronous programming, where the client of
273 the service invokes the service and continues processing immediately, without waiting for the
274 service to execute.

275 Any method with a void return type and which has no declared exceptions can be marked with a
276 **@OneWay** annotation. This means that the method is non-blocking and communication with the
277 service provider can use a binding that buffers the request and sends it at some later time.

278 For a Java client to make a non-blocking call to methods that either return values or which throw
279 exceptions, a Java client can use the JAX-WS asynchronous client API model that is described in
280 section 9. It is considered to be a best practice that service designers define one-way methods as
281 often as possible, in order to give the greatest degree of binding flexibility to deployers.

282 6.2 Callbacks

283 A **callback service** is a service that is used for **asynchronous** communication from a service
284 provider back to its client, in contrast to the communication through return values from
285 synchronous operations. Callbacks are used by **bidirectional services**, which are services that
286 have two interfaces:

- 287 • an interface for the provided service
- 288 • a callback interface that is provided by the client

289 Callbacks can be used for both remotable and local services. Either both interfaces of a
290 bidirectional service are remotable, or both are local. It is illegal to mix the two, as defined in the
291 SCA Assembly specification [SCA Assembly].

292 A callback interface is declared by using a **@Callback** annotation on a service interface, with the
293 Java Class object of the interface as a parameter. The annotation can also be applied to a method
294 or to a field of an implementation, which is used in order to have a callback injected, as explained
295 in the next section.

296 6.2.1 Using Callbacks

297 Bidirectional interfaces and callbacks are used when a simple request/response pattern isn't
298 sufficient to capture the business semantics of a service interaction. Callbacks are well suited for
299 cases when a service request can result in multiple responses or new requests from the service
300 back to the client, or where the service might respond to the client some time after the original
301 request has completed.

302 The following example shows a scenario in which bidirectional interfaces and callbacks could be
303 used. A client requests a quotation from a supplier. To process the enquiry and return the

Deleted: March

304 quotation, some suppliers might need additional information from the client. The client does not
305 know which additional items of information will be needed by different suppliers. This interaction
306 can be modeled as a bidirectional interface with callback requests to obtain the additional
307 information.

```
308 package somepackage;
309 import org.osoa.sca.annotation.Callback;
310 import org.osoa.sca.annotation.Remotable;
311 @Remotable
312 @Callback(QuotationCallback.class)
313 public interface Quotation {h
314     double requestQuotation(String productCode, int quantity);
315 }
316
317 @Remotable
318 public interface QuotationCallback {
319     String getState();
320     String getZipCode();
321     String getCreditRating();
322 }
323
```

324 In this example, the `requestQuotation` operation requests a quotation to supply a given quantity
325 of a specified product. The `QuotationCallback` interface provides a number of operations that the
326 supplier can use to obtain additional information about the client making the request. For
327 example, some suppliers might quote different prices based on the state or the zip code to which
328 the order will be shipped, and some suppliers might quote a lower price if the ordering company
329 has a good credit rating. Other suppliers might quote a standard price without requesting any
330 additional information from the client.

331 The following code snippet illustrates a possible implementation of the example service, using the
332 `@Callback` annotation to request that a callback proxy be injected.

```
333 @Callback
334 protected QuotationCallback callback;
335
336 public double requestQuotation(String productCode, int quantity) {
337     double price = getPrice(productCode, quantity);
338     double discount = 0;
339     if (quantity > 1000 && callback.getState().equals("FL")) {
340         discount = 0.05;
341     }
342     if (quantity > 10000 && callback.getCreditRating().charAt(0) == 'A') {
343         discount += 0.05;
344     }
345     return price * (1-discount);
346 }
347 }
348
```

349 The code snippet below is taken from the client of this example service. The client's service
350 implementation class implements the methods of the `QuotationCallback` interface as well as those
351 of its own service interface `ClientService`.

```
352
353 public class ClientImpl implements ClientService, QuotationCallback {
354
355     private QuotationService myService;
356
357     @Reference
358     public void setMyService(QuotationService service) {
359         myService = service;
360     }
361 }

```

```

360     }
361
362     public void aClientMethod() {
363         ...
364         double quote = myService.requestQuotation("AB123", 2000);
365         ...
366     }
367
368     public String getState() {
369         return "TX";
370     }
371     public String getZipCode() {
372         return "78746";
373     }
374     public String getCreditRating() {
375         return "AA";
376     }
377 }

```

378
379 In this example the callback is **stateless**, i.e., the callback requests do not need any information
380 relating to the original service request. For a callback that needs information relating to the
381 original service request (a **stateful** callback), this information can be passed to the client by the
382 service provider as parameters on the callback request.

383 6.2.2 Callback Instance Management

384 Instance management for callback requests received by the client of the bidirectional service is
385 handled in the same way as instance management for regular service requests. If the client
386 implementation has STATELESS scope, the callback is dispatched using a newly initialized
387 instance. If the client implementation has COMPOSITE scope, the callback is dispatched using the
388 same shared instance that is used to dispatch regular service requests.

389 As described in section 6.7.1, a stateful callback can obtain information relating to the original
390 service request from parameters on the callback request. Alternatively, a composite-scoped client
391 could store information relating to the original request as instance data and retrieve it when the
392 callback request is received. These approaches could be combined by using a key passed on the
393 callback request (e.g., an order ID) to retrieve information that was stored in a composite-scoped
394 instance by the client code that made the original request.

395 6.2.3 Implementing Multiple Bidirectional Interfaces

396 Since it is possible for a single implementation class to implement multiple services, it is also
397 possible for callbacks to be defined for each of the services that it implements. The service
398 implementation can include an injected field for each of its callbacks. The runtime injects the
399 callback onto the appropriate field based on the type of the callback. The following shows the
400 declaration of two fields, each of which corresponds to a particular service offered by the
401 implementation.

```

402 @Callback
403 protected MyService1Callback callback1;
404
405 @Callback
406 protected MyService2Callback callback2;

```

408
409 If a single callback has a type that is compatible with multiple declared callback fields, then all of
410 them will be set.

Deleted: March

411 6.2.4 Accessing Callbacks

412 In addition to injecting a reference to a callback service, it is also possible to obtain a reference to
413 a Callback instance by annotating a field or method of type **ServiceReference** with the
414 **@Callback** annotation.

415
416 A reference implementing the callback service interface can be obtained using
417 `ServiceReference.getService()`.

418 The following example fragments come from a service implementation that uses the callback API:

```
419 @Callback
420 protected ServiceReference<MyCallback> callback;
421
422 public void someMethod() {
423     MyCallback myCallback = callback.getCallback();    ...
424
425     myCallback.receiveResult(theResult);
426 }
427
428
429
```

430 Because ServiceReference objects are serializable, they can be stored persistently and retrieved at
431 a later time to make a callback invocation after the associated service request has completed.
432 ServiceReference objects can also be passed as parameters on service invocations, enabling the
433 responsibility for making the callback to be delegated to another service.

434 Alternatively, a callback can be retrieved programmatically using the **RequestContext** API. The
435 snippet below shows how to retrieve a callback in a method programmatically:

```
436 public void someMethod() {
437     MyCallback myCallback =
438         ComponentContext.getRequestContext().getCallback();
439     ...
440
441     myCallback.receiveResult(theResult);
442 }
443
444
445
```

446 On the client side, the service that implements the callback can access the callback ID that was
447 returned with the callback operation by accessing the request context, as follows:

```
448 @Context
449 protected RequestContext requestContext;
450
451 void receiveResult(Object theResult) {
452     Object refParams =
453         requestContext.getServiceReference().getCallbackID();
454     ...
455 }
456
```

457
458 This is necessary if the service implementation has COMPOSITE scope, because callback injection
459 is not performed for composite-scoped implementations.

460 7 Policy Annotations for Java

461 SCA provides facilities for the attachment of policy-related metadata to SCA assemblies, which
462 influence how implementations, services and references behave at runtime. The policy facilities
463 are described in [the SCA Policy Framework specification \[POLICY\]](#). In particular, the facilities
464 include Intents and Policy Sets, where intents express abstract, high-level policy requirements and
465 policy sets express low-level detailed concrete policies.

466 Policy metadata can be added to SCA assemblies through the means of declarative statements
467 placed into Composite documents and into Component Type documents. These annotations are
468 completely independent of implementation code, allowing policy to be applied during the assembly
469 and deployment phases of application development.

470 However, it can be useful and more natural to attach policy metadata directly to the code of
471 implementations. This is particularly important where the policies concerned are relied on by the
472 code itself. An example of this from the Security domain is where the implementation code
473 expects to run under a specific security Role and where any service operations invoked on the
474 implementation have to be authorized to ensure that the client has the correct rights to use the
475 operations concerned. By annotating the code with appropriate policy metadata, the developer
476 can rest assured that this metadata is not lost or forgotten during the assembly and deployment
477 phases.

478 This specification has a series of annotations which provide the capability for the developer to
479 attach policy information to Java implementation code. The annotations concerned first provide
480 general facilities for attaching SCA Intents and Policy Sets to Java code. Secondly, there are
481 further specific annotations that deal with particular policy intents for certain policy domains such
482 as Security.

483 This specification supports using [the Common Annotation for Java Platform specification \(JSR-250\)](#)
484 [\[JSR-250\]](#). An implication of adopting the common annotation for Java platform specification is
485 that the SCA Java specification supports consistent annotation and Java class inheritance
486 relationships.

487 7.1 General Intent Annotations

488 SCA provides the annotation `@Requires` for the attachment of any intent to a Java class, to a
489 Java interface or to elements within classes and interfaces such as methods and fields.

490 The `@Requires` annotation can attach one or multiple intents in a single statement.

491 Each intent is expressed as a string. Intents are XML QNames, which consist of a Namespace URI
492 followed by the name of the Intent. The precise form used follows the string representation used
493 by the `javax.xml.namespace.QName` class, which is as follows:

```
494     "{ " + Namespace URI + " } " + intentname
```

495 Intents can be qualified, in which case the string consists of the base intent name, followed by a
496 ".", followed by the name of the qualifier. There can also be multiple levels of qualification.

497 This representation is quite verbose, so we expect that reusable String constants will be defined
498 for the namespace part of this string, as well as for each intent that is used by Java code. SCA
499 defines constants for intents such as the following:

```
500     public static final String SCA_PREFIX=  
501         "{http://docs.oasis-open.org/ns/opencsa/sca/200712}";  
502     public static final String CONFIDENTIALITY = SCA_PREFIX + "confidentiality";  
503     public static final String CONFIDENTIALITY_MESSAGE = CONFIDENTIALITY + ".message";  
504
```

505 Notice that, by convention, qualified intents include the qualifier as part of the name of the
506 constant, separated by an underscore. These intent constants are defined in the file that defines

Deleted: March

507 an annotation for the intent (annotations for intents, and the formal definition of these constants,
508 are covered in a following section).

509 Multiple intents (qualified or not) are expressed as separate strings within an array declaration.

510 An example of the @Requires annotation with 2 qualified intents (from the Security domain)
511 follows:

```
512     @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE})
```

513

514 This attaches the intents "confidentiality.message" and "integrity.message".

515 The following is an example of a reference requiring support for confidentiality:

```
516     package com.foo;
517
518     import static org.oasisopen.sca.annotation.Confidentiality.*;
519     import static org.oasisopen.sca.annotation.Reference;
520     import static org.oasisopen.sca.annotation.Requires;
521
522     public class Foo {
523         @Requires(CONFIDENTIALITY)
524         @Reference
525         public void setBar(Bar bar) {
526             ...
527         }
528     }
529
```

530 Users can also choose to only use constants for the namespace part of the QName, so that they
531 can add new intents without having to define new constants. In that case, this definition would
532 instead look like this:

```
533     package com.foo;
534
535     import static org.oasisopen.sca.Constants.*;
536     import static org.oasisopen.sca.annotation.Reference;
537     import static org.oasisopen.sca.annotation.Requires;
538
539     public class Foo {
540         @Requires(SCA_PREFIX+"confidentiality")
541         @Reference
542         public void setBar(Bar bar) {
543             ...
544         }
545     }
546
```

547 The formal syntax [EBNF-Syntax] for the @Requires annotation follows:

```
548 '@Requires("'" QualifiedIntent "'" (','" QualifiedIntent "'"*)* ''')
```

549 where

```
550     QualifiedIntent ::= QName('.' Qualifier)*
551     Qualifier ::= NCName
```

552

553 See [section @Requires](#) for the formal definition of the @Requires annotation.

554 **7.2 Specific Intent Annotations**

555 In addition to the general intent annotation supplied by the @Requires annotation described
556 above, it is also possible to have Java annotations that correspond to specific policy intents. SCA
557 provides a number of these specific intent annotations and it is also possible to create new specific
558 intent annotations for any intent.

559 The general form of these specific intent annotations is an annotation with a name derived from
560 the name of the intent itself. If the intent is a qualified intent, qualifiers are supplied as an
561 attribute to the annotation in the form of a string or an array of strings.

562 For example, the SCA confidentiality intent described in [the section on General Intent Annotations](#)
563 using the @Requires(CONFIDENTIALITY) annotation can also be specified with the
564 @Confidentiality specific intent annotation. The specific intent annotation for the "integrity"
565 security intent is:

566 @Integrity

567 An example of a qualified specific intent for the "authentication" intent is:

568 @Authentication({ "message", "transport" })

569 This annotation attaches the pair of qualified intents: "authentication.message" and
570 "authentication.transport" (the sca: namespace is assumed in this both of these cases –
571 "http://docs.oasis-open.org/ns/opencsa/sca/200712").

572 The general form of specific intent annotations is:

573 '@' Intent ('(' qualifiers ')')?

574 where Intent is an NCName that denotes a particular type of intent.

575 Intent ::= NCName
576 qualifiers ::= "" qualifier "" (',' qualifier "")*
577 qualifier ::= NCName ('.' qualifier)?
578

579 **7.2.1 How to Create Specific Intent Annotations**

580 **SCA identifies annotations that correspond to intents by providing an @Intent annotation which**
581 **MUST be used in the definition of a specific intent annotation. [JCA70001]**

582 The @Intent annotation takes a single parameter, which (like the @Requires annotation) is the
583 String form of the QName of the intent. As part of the intent definition, it is good practice
584 (although not required) to also create String constants for the Namespace, for the Intent and for
585 Qualified versions of the Intent (if defined). These String constants are then available for use with
586 the @Requires annotation and it is also possible to use one or more of them as parameters to the
587 specific intent annotation.

588 Alternatively, the QName of the intent can be specified using separate parameters for the
589 targetNamespace and the localPart, for example:

590 @Intent(targetNamespace=SCA_NS, localPart="confidentiality").

591 See [section @Intent](#) for the formal definition of the @Intent annotation.

592 When an intent can be qualified, it is good practice for the first attribute of the annotation to be a
593 string (or an array of strings) which holds one or more qualifiers.

594 In this case, the attribute's definition needs to be marked with the @Qualifier annotation. The
595 @Qualifier tells SCA that the value of the attribute is treated as a qualifier for the intent
596 represented by the whole annotation. If more than one qualifier value is specified in an
597 annotation, it means that multiple qualified forms exist. For example:

598 @Confidentiality({ "message", "transport" })

599 implies that both of the qualified intents "confidentiality.message" and "confidentiality.transport"
600 are set for the element to which the @confidentiality annotation is attached.

Deleted: SCA identifies annotations that correspond to intents by providing an @Intent annotation which MUST be used in the definition of a specific intent annotation.

Deleted: SCA identifies annotations that correspond to intents by providing an @Intent annotation which MUST be used in the definition of a specific intent annotation.

Deleted: March

601 See [section @Qualifier](#) for the formal definition of the @Qualifier annotation.

602 Examples of the use of the @Intent and the @Qualifier annotations in the definition of specific
603 intent annotations are shown in [the section dealing with Security Interaction Policy](#).

604 7.3 Application of Intent Annotations

605 The SCA Intent annotations can be applied to the following Java elements:

- 606 • Java class
- 607 • Java interface
- 608 • Method
- 609 • Field
- 610 • Constructor parameter

611 Where multiple intent annotations (general or specific) are applied to the same Java element, they
612 are additive in effect. An example of multiple policy annotations being used together follows:

```
613 @Authentication  
614 @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE})
```

615 In this case, the effective intents are "authentication", "confidentiality.message" and
616 "integrity.message".

617 If an annotation is specified at both the class/interface level and the method or field level, then
618 the method or field level annotation completely overrides the class level annotation of the same
619 base intent name.

620 The intent annotation can be applied either to classes or to class methods when adding annotated
621 policy on SCA services. Applying an intent to the setter method in a reference injection approach
622 allows intents to be defined at references.

623 7.3.1 Inheritance And Annotation

624 The inheritance rules for annotations are consistent with the common annotation specification, JSR
625 250 [JSR-250]

626 The following example shows the inheritance relations of intents on classes, operations, and super
627 classes.

```
628 package services.hello;  
629 import org.oasisopen.sca.annotation.Remotable;  
630 import org.oasisopen.sca.annotation.Integrity;  
631 import org.oasisopen.sca.annotation.Authentication;  
632  
633 @Integrity("transport")  
634 @Authentication  
635 public class HelloService {  
636     @Integrity  
637     @Authentication("message")  
638     public String hello(String message) {...}  
639  
640     @Integrity  
641     @Authentication("transport")  
642     public String helloThere() {...}  
643 }  
644  
645 package services.hello;  
646 import org.oasisopen.sca.annotation.Remotable;  
647 import org.oasisopen.sca.annotation.Confidentiality;  
648 import org.oasisopen.sca.annotation.Authentication;
```

```

649
650     @Confidentiality("message")
651     public class HelloChildService extends HelloService {
652         @Confidentiality("transport")
653         public String hello(String message) {...}
654         @Authentication
655         String helloWorld() {...}
656     }

```

657 Example 2a. Usage example of annotated policy and inheritance.

658
659 The effective intent annotation on the *helloWorld* method of the *HelloChildService* is
660 Integrity("transport"), @Authentication, and @Confidentiality("message").

661 The effective intent annotation on the *hello* method of the *HelloChildService* is
662 @Integrity("transport"), @Authentication, and @Confidentiality("transport"),

663 The effective intent annotation on the *helloThere* method of the *HelloChildService* is @Integrity
664 and @Authentication("transport"), the same as in *HelloService* class.

665 The effective intent annotation on the *hello* method of the *HelloService* is @Integrity and
666 @Authentication("message")

667
668 The listing below contains the equivalent declarative security interaction policy of the HelloService
669 and HelloChildService implementation corresponding to the Java interfaces and classes shown in
670 Example 2a.

```

671
672 <?xml version="1.0" encoding="ASCII"?>
673 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
674           name="HelloServiceComposite" >
675     <service name="HelloService" requires="integrity/transport
676           authentication">
677       ...
678     </service>
679     <service name="HelloChildService" requires="integrity/transport
680           authentication confidentiality/message">
681       ...
682     </service>
683     ...
684
685     <component name="HelloServiceComponent">*
686       <implementation.java class="services.hello.HelloService"/>
687       <operation name="hello" requires="integrity
688           authentication/message"/>
689       <operation name="helloThere"
690           requires="integrity
691           authentication/transport"/>
692     </component>
693     <component name="HelloChildServiceComponent">*
694       <implementation.java
695           class="services.hello.HelloChildService" />
696       <operation name="hello"
697           requires="confidentiality/transport"/>
698       <operation name="helloThere" requires=" integrity/transport
699           authentication"/>
700       <operation name="helloWorld" requires="authentication"/>
701     </component>
702

```

Deleted: March

703 ...
704
705 </composite>
706

707 Example 2b. Declaratives intents equivalent to annotated intents in Example 2a.
708

709 7.4 Relationship of Declarative And Annotated Intents

710 Annotated intents on a Java class cannot be overridden by declarative intents in a composite
711 document which uses the class as an implementation. This rule follows the general rule for intents
712 that they represent requirements of an implementation in the form of a restriction that cannot be
713 relaxed.

714 However, a restriction can be made more restrictive so that an unqualified version of an intent
715 expressed through an annotation in the Java class can be qualified by a declarative intent in a
716 using composite document.

717 7.5 Policy Set Annotations

718 The SCA Policy Framework uses Policy Sets to capture detailed low-level concrete policies. For
719 example, a concrete policy is the specific encryption algorithm to use when encrypting messages
720 when using a specific communication protocol to link a reference to a service.

721 Policy Sets can be applied directly to Java implementations using the **@PolicySets** annotation.
722 The @PolicySets annotation either takes the QName of a single policy set as a string or the name
723 of two or more policy sets as an array of strings:
724

```
725       @PolicySets( "<policy set QName>" (, "<policy set QName>")* )
```

726

727 As for intents, PolicySet names are QNames – in the form of "{Namespace-URI}localPart".

728 An example of the @PolicySets annotation:

729

```
730       @Reference(name="helloService", required=true)  
731       @PolicySets({ MY_NS + "WS_Encryption_Policy",  
732                   MY_NS + "WS_Authentication_Policy" })  
733       public setHelloService(HelloService service) {  
734           ...  
735       }  
736
```

737 In this case, the Policy Sets WS_Encryption_Policy and WS_Authentication_Policy are applied, both
738 using the namespace defined for the constant MY_NS.

739 PolicySets need to satisfy intents expressed for the implementation when both are present,
740 according to the rules defined in [the Policy Framework specification \[POLICY\]](#).

741 The SCA Policy Set annotation can be applied to the following Java elements:

- 742 • Java class
- 743 • Java interface
- 744 • Method
- 745 • Field
- 746 • Constructor parameter

747 7.6 Security Policy Annotations

748 This section introduces annotations for SCA's security intents, as defined in [the SCA Policy](#)
749 [Framework specification \[POLICY\]](#).

750 7.6.1 Security Interaction Policy

751 The following interaction policy Intents and qualifiers are defined for Security Policy, which apply
752 to the operation of services and references of an implementation:

- 753 • @Integrity
- 754 • @Confidentiality
- 755 • @Authentication

756 All three of these intents have the same pair of Qualifiers:

- 757 • message
- 758 • transport

759 The formal definitions of the @Authentication, @Confidentiality and @Integrity annotations are
760 found in the sections [@Authentication](#), [@Confidentiality](#) and [@Integrity](#).

761 The following example shows an example of applying an intent to the setter method used to inject
762 a reference. Accessing the hello operation of the referenced HelloService requires both
763 "integrity.message" and "authentication.message" intents to be honored.

```
764  
765 package services.hello;  
766 //Interface for HelloService  
767 public interface HelloService {  
768     String hello(String helloMsg);  
769 }  
770  
771 package services.client;  
772 // Interface for ClientService  
773 public interface ClientService {  
774     public void clientMethod();  
775 }  
776  
777 // Implementation class for ClientService  
778 package services.client;  
779  
780 import services.hello.HelloService;  
781 import org.oasisopen.sca.annotation.*;  
782  
783 @Service(ClientService.class)  
784 public class ClientServiceImpl implements ClientService {  
785  
786     private HelloService helloService;  
787  
788     @Reference(name="helloService", required=true)  
789     @Integrity("message")  
790     @Authentication("message")  
791     public void setHelloService(HelloService service) {  
792         helloService = service;  
793     }  
794  
795     public void clientMethod() {  
796         String result = helloService.hello("Hello World!");
```

Deleted: March

```
797     ...
798     }
799 }
```

800
801 Example 1. Usage of annotated intents on a reference.

802 7.6.2 Security Implementation Policy

803 SCA defines a number of security policy annotations that apply as policies to implementations
804 themselves. These annotations mostly have to do with authorization and security identity. The
805 following authorization and security identity annotations (as defined in JSR 250) are supported:

- 806 • RunAs
807 Takes as a parameter a string which is the name of a Security role.
808 eg. @RunAs("Manager") Code marked with this annotation executes with the Security
809 permissions of the identified role.
810
- 811 • RolesAllowed
812 Takes as a parameter a single string or an array of strings which represent one or more
813 role names. When present, the implementation can only be accessed by principals whose
814 role corresponds to one of the role names listed in the @roles attribute. How role names
815 are mapped to security principals is implementation dependent (SCA does not define this).
816 eg. @RolesAllowed({"Manager", "Employee"})
817
- 818 • PermitAll
819 No parameters. When present, grants access to all roles.
820
- 821 • DenyAll
822 No parameters. When present, denies access to all roles.
823
- 824 • DeclareRoles
825 Takes as a parameter a string or an array of strings which identify one or more role names
826 that form the set of roles used by the implementation.
827 eg. @DeclareRoles({"Manager", "Employee", "Customer"})

828 (all these are declared in the Java package javax.annotation.security)

829 For a full explanation of these intents, see [the Policy Framework specification \[POLICY\]](#).

830 7.6.2.1 Annotated Implementation Policy Example

831 The following is an example showing annotated security implementation policy:

```
832
833 package services.account;
834 @Remotable
835 public interface AccountService {
836     AccountReport getAccountReport(String customerID);
837     float fromUSDollarToCurrency(float value);
838 }
```

839
840 The following is a full listing of the AccountServiceImpl class, showing the Service it implements,
841 plus the service references it makes and the settable properties that it has, along with a set of
842 implementation policy annotations:

```
843
844 package services.account;
```

```

845     import java.util.List;
846     import commonj.sdo.DataFactory;
847     import org.oasisopen.sca.annotation.Property;
848     import org.oasisopen.sca.annotation.Reference;
849     import org.oasisopen.sca.annotation.RolesAllowed;
850     import org.oasisopen.sca.annotation.RunAs;
851     import org.oasisopen.sca.annotation.PermitAll;
852     import services.accountdata.AccountDataService;
853     import services.accountdata.CheckingAccount;
854     import services.accountdata.SavingsAccount;
855     import services.accountdata.StockAccount;
856     import services.stockquote.StockQuoteService;
857     @RolesAllowed("customers")
858     @RunAs("accountants" )
859     public class AccountServiceImpl implements AccountService {
860
861         @Property
862         protected String currency = "USD";
863
864         @Reference
865         protected AccountDataService accountDataService;
866         @Reference
867         protected StockQuoteService stockQuoteService;
868
869         @RolesAllowed({"customers", "accountants"})
870         public AccountReport getAccountReport(String customerID) {
871
872             DataFactory dataFactory = DataFactory.INSTANCE;
873             AccountReport accountReport =
874                 (AccountReport)dataFactory.create(AccountReport.class);
875             List accountSummaries = accountReport.getAccountSummaries();
876
877             CheckingAccount checkingAccount =
878                 accountDataService.getCheckingAccount(customerID);
879             AccountSummary checkingAccountSummary =
880                 (AccountSummary)dataFactory.create(AccountSummary.class);
881             checkingAccountSummary.setAccountNumber(checkingAccount.getAccountNumber()
882 );
883             checkingAccountSummary.setAccountType("checking");
884             checkingAccountSummary.setBalance(fromUSDollarToCurrency
885                 (checkingAccount.getBalance()));
886             accountSummaries.add(checkingAccountSummary);
887
888             SavingsAccount savingsAccount =
889                 accountDataService.getSavingsAccount(customerID);
890             AccountSummary savingsAccountSummary =
891                 (AccountSummary)dataFactory.create(AccountSummary.class);
892             savingsAccountSummary.setAccountNumber(savingsAccount.getAccountNumber());
893             savingsAccountSummary.setAccountType("savings");
894             savingsAccountSummary.setBalance(fromUSDollarToCurrency
895                 (savingsAccount.getBalance()));
896             accountSummaries.add(savingsAccountSummary);
897
898             StockAccount stockAccount =
899                 accountDataService.getStockAccount(customerID);
900             AccountSummary stockAccountSummary =

```

Deleted: March

```

903         (AccountSummary)dataFactory.create(AccountSummary.class);
904     stockAccountSummary.setAccountNumber(stockAccount.getAccountNumber());
905     stockAccountSummary.setAccountType("stock");
906     float balance= (stockQuoteService.getQuote(stockAccount.getSymbol()))*
907         stockAccount.getQuantity();
908     stockAccountSummary.setBalance(fromUSDollarToCurrency(balance));
909     accountSummaries.add(stockAccountSummary);
910
911     return accountReport;
912 }
913
914 @PermitAll
915 public float fromUSDollarToCurrency(float value) {
916
917     if (currency.equals("USD")) return value;
918     if (currency.equals("EURO")) return value * 0.8f;
919     return 0.0f;
920 }
921 }

```

922 Example 3. Usage of annotated security implementation policy for the java language.

923 In this example, the implementation class as a whole is marked:

- 924 • @RolesAllowed("customers") - indicating that customers have access to the
- 925 implementation as a whole
- 926 • @RunAs("accountants") - indicating that the code in the implementation runs with the
- 927 permissions of accountants

928 The getAccountReport(..) method is marked with @RolesAllowed({"customers", "accountants"}),

929 which indicates that this method can be called by both customers and accountants.

930 The fromUSDollarToCurrency() method is marked with @PermitAll, which means that this method

931 can be called by any role.

932 8 Java API

933 This section provides a reference for the Java API offered by SCA.

934 8.1 Component Context

935 The following Java code defines the **ComponentContext** interface:

```
936
937 package org.oasisopen.sca;
938
939 public interface ComponentContext {
940     String getURI();
941
942     <B> B getService(Class<B> businessInterface, String referenceName);
943
944     <B> ServiceReference<B> getServiceReference(Class<B> businessInterface,
945                                             String referenceName);
946
947     <B> Collection<B> getServices(Class<B> businessInterface,
948                               String referenceName);
949
950     <B> Collection<ServiceReference<B>> getServiceReferences(Class<B>
951                                                         businessInterface, String referenceName);
952
953     <B> ServiceReference<B> createSelfReference(Class<B>
954                                             businessInterface);
955
956     <B> ServiceReference<B> createSelfReference(Class<B> businessInterface,
957                                             String serviceName);
958
959     <B> B getProperty(Class<B> type, String propertyName);
960
961     <B, R extends ServiceReference<B>> R cast(B target)
962         throws IllegalArgumentException;
963
964     RequestContext getRequestContext();
965
966
967 }
```

- 968
- 969 • **getURI()** - returns the absolute URI of the component within the SCA domain
 - 970 • **getService(Class businessInterface, String referenceName)** - Returns a proxy for
971 the reference defined by the current component. The getService() method takes as its
972 input arguments the Java type used to represent the target service on the client and the
973 name of the service reference. It returns an object providing access to the service. The
974 returned object implements the Java interface the service is typed with.
975 **ComponentContext.getService method MUST throw an IllegalArgumentException if the**
976 **reference identified by the referenceName parameter has multiplicity of 0..n or**
977 **1..n.[JCA80001]**
 - 978 • **getServiceReference(Class businessInterface, String referenceName)** - Returns a
979 ServiceReference defined by the current component. This method MUST throw an
980 IllegalArgumentException if the reference has multiplicity greater than one.

Deleted: March

- 981 • **getServices(Class businessInterface, String referenceName)** – Returns a list of
982 typed service proxies for a business interface type and a reference name.
- 983 • **getServiceReferences(Class businessInterface, String referenceName)** –Returns a
984 list typed service references for a business interface type and a reference name.
- 985 • **createSelfReference(Class businessInterface)** – Returns a ServiceReference that can
986 be used to invoke this component over the designated service.
- 987 • **createSelfReference(Class businessInterface, String serviceName)** – Returns a
988 ServiceReference that can be used to invoke this component over the designated service.
989 Service name explicitly declares the service name to invoke
- 990 • **getProperty (Class type, String propertyName)** - Returns the value of an SCA
991 property defined by this component.
- 992 • **getRequestContext()** - Returns the context for the current SCA service request, or null if
993 there is no current request or if the context is unavailable. **The**
994 **ComponentContext.getRequestContext** method **MUST** return non-null when invoked during
995 the execution of a Java business method for a service operation or a callback operation, **on**
996 **the same thread that the SCA runtime provided, and MUST** return null in all other cases.
997 **[JCA80002]**
- 998 • **cast(B target)** - Casts a type-safe reference to a ServiceReference

999 A component can access its component context by defining a field or setter method typed by
1000 **org.oasisopen.sca.ComponentContext** and annotated with **@Context**. To access the target
1001 service, the component uses **ComponentContext.getService(..)**.

1002 The following shows an example of component context usage in a Java class using the @Context
1003 annotation.

```
1004 private ComponentContext componentContext;
1005
1006 @Context
1007 public void setContext(ComponentContext context) {
1008     componentContext = context;
1009 }
1010
1011 public void doSomething() {
1012     HelloWorld service =
1013     componentContext.getService(HelloWorld.class, "HelloWorldComponent");
1014     service.hello("hello");
1015 }
```

1016 8.2 Request Context

1017 The following shows the **RequestContext** interface:

```
1018 package org.oasisopen.sca;
1019
1020 import javax.security.auth.Subject;
1021
1022 public interface RequestContext {
1023
1024     Subject getSecuritySubject();
1025
1026     String getServiceName();
1027     <CB> ServiceReference<CB> getCallbackReference();
1028     <CB> CB getCallback();
1029     <B> ServiceReference<B> getServiceReference();
1030 }
```

Deleted: ¶
Similarly, non-SCA client code can use the ComponentContext API to perform operations against a component in an SCA domain. How the non-SCA client code obtains a reference to a ComponentContext is runtime specific. ¶

Formatted: Bullets and Numbering

Deleted: March

1031 }
1032 }
1033 }

1034 The RequestContext interface has the following methods:

- 1035 • **getSecuritySubject()** – Returns the JAAS Subject of the current request
- 1036 • **getServiceName()** – Returns the name of the service on the Java implementation the
1037 request came in on
- 1038 • **getCallbackReference()** – Returns a service reference to the callback as specified by the
1039 caller. This method returns null when called for a service request whose interface is not
1040 bidirectional or when called for a callback request.
- 1041 • **getCallback()** – Returns a proxy for the callback as specified by the caller. Similar to the
1042 getCallbackReference() method, this method returns null when called for a service request
1043 whose interface is not bidirectional or when called for a callback request.
- 1044 • **getServiceReference()** – When invoked during the execution of a service operation, the
1045 getServiceReference method MUST return a ServiceReference that represents the service
1046 that was invoked. When invoked during the execution of a callback operation, the
1047 getServiceReference method MUST return a ServiceReference that represents the callback
1048 that was invoked. [JCA80003]

Comment [ME2]: Need a reference to JAAS here

Comment [ME3]: What happens if there is no JAAS subject?

Deleted: When invoked during the execution of a service operation, the getServiceReference method MUST return a ServiceReference that represents the service that was invoked. When invoked during the execution of a callback operation, the getServiceReference method MUST return a ServiceReference that represents the callback that was invoked.

Deleted: When invoked during the execution of a service operation, the getServiceReference method MUST return a ServiceReference that represents the service that was invoked. When invoked during the execution of a callback operation, the getServiceReference method MUST return a ServiceReference that represents the callback that was invoked.

1049 8.3 ServiceReference

1050 ServiceReferences can be injected using the @Reference annotation on a field, a setter method,
1051 or constructor parameter taking the type ServiceReference. The detailed description of the usage
1052 of these methods is described in the section on Asynchronous Programming in this document.

1053 The following Java code defines the **ServiceReference** interface:

```
1054 package org.oasisopen.sca;  
1055  
1056 public interface ServiceReference<B> extends java.io.Serializable {  
1057     B getService();  
1058     Class<B> getBusinessInterface();  
1060 }  
1061
```

1062 The ServiceReference interface has the following methods:

- 1063 • **getService()** - Returns a type-safe reference to the target of this reference. The instance
1064 returned is guaranteed to implement the business interface for this reference. The value
1065 returned is a proxy to the target that implements the business interface associated with this
1066 reference.
- 1067 • **getBusinessInterface()** – Returns the Java class for the business interface associated with
1068 this reference.

1069 8.4 ServiceRuntimeException

1070 The following snippet shows the **ServiceRuntimeException**.

```
1071  
1072 package org.oasisopen.sca;  
1073  
1074 public class ServiceRuntimeException extends RuntimeException {  
1075     ...  
1076 }  
1077
```

1078 This exception signals problems in the management of SCA component execution.

Deleted: March

1079 8.5 ServiceUnavailableException

1080 The following snippet shows the *ServiceUnavailableException*.

```
1081 package org.oasisopen.sca;
1082
1083
1084 public class ServiceUnavailableException extends ServiceRuntimeException {
1085     ...
1086 }
1087
```

1088 This exception signals problems in the interaction with remote services. These are exceptions
1089 that can be transient, so retrying is appropriate. Any exception that is a ServiceRuntimeException
1090 that is *not* a ServiceUnavailableException is unlikely to be resolved by retrying the operation, since
1091 it most likely requires human intervention

1092 8.6 InvalidServiceException

1093 The following snippet shows the *InvalidServiceException*.

```
1094 package org.oasisopen.sca;
1095
1096
1097 public class InvalidServiceException extends ServiceRuntimeException {
1098     ...
1099 }
1100
```

1101 This exception signals that the ServiceReference is no longer valid. This can happen when the
1102 target of the reference is undeployed. This exception is not transient and therefore is unlikely to
1103 be resolved by retrying the operation and will most likely require human intervention.

1104 8.7 Constants

1105 The SCA *Constants* interface defines a number of constant values that are used in the SCA Java
1106 APIs and Annotations. The following snippet shows the Constants interface:

```
1107 package org.oasisopen.sca;
1108
1109 public interface Constants {
1110     String SCA_NS="http://docs.oasis-open.org/ns/opencsa/sca/200712";
1111     String SCA_PREFIX = "{ "+SCA_NS+"}";
1112 }
1113
```

1114 8.8 SCAClient Interface

1115 [The SCAClient interface can be used by client code to obtain a proxy reference object for a service](#)
1116 [within an SCA Domain, through which the client code can invoke operations of that service. This](#)
1117 [is particularly useful for client code that is running outside the SCA Domain containing the target](#)
1118 [service, for example where the code is "unmanaged" and is not running under an SCA runtime.](#)

1119 [The following shows the SCAClient interface:](#)

```
1120 package org.oasisopen.sca.client;
1121
1122 public interface SCAClient {
1123
1124     <T> T getService(Class<T> interfaze,
1125                   String serviceURI)
1126     throws NoSuchServiceException, NoSuchDomainException;

```

1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144

1145
1146
1147
1148

1149
1150
1151

1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176

```
}  
  
getService method:  
  
Obtains a proxy reference object for a specified target service in a specified SCA Domain.  
  
Returns:  


- proxy object which implements the business interface T  
Invocations of a business method of the proxy causes the invocation of the corresponding operation of the target service .

  
Parameters:  


- interfaze - a Java interface class which is the business interface of the target service
- serviceURI - a String containing the relative URI of the target service within its SCA Domain.  
Takes the form componentName/serviceName or can also take the extended form componentName/serviceName/bindingName to use a specific binding of the target service

  
Exceptions:  


- NoSuchServiceException - thrown if a service with the relative URI serviceURI and a business interface which matches interfaze cannot be found in the SCA Domain targeted by the SCAClient object

```

8.9 SCAClientFactory Class

The `SCAClientFactory` class provides the means for client code to obtain an object which implements the `SCAClient` interface which is used in turn to obtain a proxy reference object to a service within an SCA Domain.

The `SCAClientFactory` is an abstract class which provides a set of static `newInstance(...)` methods which the client can invoke in order to obtain a object implementing the `SCAClient` interface for a particular SCA Domain.

The `SCAClientFactory` class is as follows:

```
package org.oasisopen.sca.client;  
  
public abstract class SCAClientFactory {  
    protected static SCAClientFactoryFinder factoryFinder;  
    private URI domainURI;  
  
    private SCAClientFactory() {}  
    protected SCAClientFactory(URI domainURI) {...}  
    protected URI getDomainURI() {...}  
    public static SCAClient newInstance( URI domainURI ) {...}  
    public static SCAClient newInstance(Properties properties,  
                                         URI domainURI) {...}  
    public static SCAClient newInstance(ClassLoader classLoader,  
                                         URI domainURI) {...}
```

Formatted: Indent: Before: 0.25", First line: 0.25"

Deleted: ¶

Formatted: Indent: Before: 0.5"

Formatted: Indent: Before: 0.5"

Deleted: March

```
1177     public static SCAClient newInstance(Properties properties,  
1178                                     ClassLoader classLoader,  
1179                                     URI domainURI) {...}  
1180  
1181     protected abstract SCAClient createSCAClient();  
1182 }  
1183
```

1184 **newInstance (URI) method:**

1185 Obtains a object implementing the SCAClient interface.

1186 Returns:

- 1187 • **object** which implements the SCAClient interface

1188 Parameters:

- 1189 • **domainURI** - a URI for the SCA Domain which is targeted by the returned SCAClient
1190 object

1191 Exceptions:

- 1192 • **none**

1193
1194 **newInstance(Properties, URI) method:**

1195 Obtains a object implementing the SCAClient interface, using a specified set of properties.

1196 Returns:

- 1197 • **object** which implements the SCAClient interface

1198 Parameters:

- 1199 • **properties** - a set of Properties that can be used when creating the object which
1200 implements the SCAClient interface.
- 1201 • **domainURI** - a URI for the SCA Domain which is targeted by the returned SCAClient
1202 object

1203 Exceptions:

- 1204 • **none**

1205
1206 **newInstance(Classloader, URIR) method:**

1207 Obtains a object implementing the SCAClient interface using a specified classloader.

1208 Returns:

- 1209 • **object** which implements the SCAClient interface

1210 Parameters:

- 1211 • **classLoader** - a ClassLoader to use when creating the object which implements the
1212 SCAClient interface.
- 1213 • **domainURI** - a URI for the SCA Domain which is targeted by the returned SCAClient
1214 object

1215 Exceptions:

- 1216 • **none**

1217
1218 **newInstance(Properties, Classloader, URI) method:**

← - - - Formatted: Bullets and
Numbering

← - - - Formatted: Bullets and
Numbering

Deleted: March

1219 Obtains a [object](#) implementing the [SCAClient](#) interface using a specified set of properties and a
1220 [specified classloader](#).

1221 [Returns:](#)

- 1222 • [object](#) which implements the [SCAClient](#) interface

1223 [Parameters:](#)

- 1224 • [properties](#) - a set of [Properties](#) that can be used when creating the object which
1225 [implements the SCAClient interface](#).
- 1226 • [classLoader](#) - a [ClassLoader](#) to use when creating the object which implements the
1227 [SCAClient interface](#).
- 1228 • [domainURI](#) - a [URI](#) for the [SCA Domain](#) which is targeted by the returned [SCAClient](#)
1229 [object](#)

1230 [Exceptions:](#)

- 1231 • [none](#)

1232

1233 [SCAClientFactory \(URI \) method:](#) a [single argument constructor that must be available on all](#)
1234 [concrete subclasses of SCAClientFactory](#). The [URI](#) required is the [URI](#) of the [Domain](#) targeted by
1235 [the SCAClientFactory](#)

1236 [getDomainURI\(\) method:](#)

1237 Obtains the [Domain URI](#) value - for use by concrete subclasses of [SCAClientFactory](#)

1238 [Returns:](#)

- 1239 • [URI](#) of the target [SCA Domain](#) for this [SCAClientFactory](#)

1240 [Parameters:](#)

- 1241 • [none](#)

1242 [Exceptions:](#)

- 1243 • [none](#)

1244

1245 [defaultFactory](#) [protected field:](#)

1246 Provides a means by which a provider of an [SCAClientFactory](#) implementation can inject a [factory](#)
1247 [finder implementation into the abstract SCAClientFactory class - once this is done, future](#)
1248 [invocations of the SCAClientFactory use the injected factory finder to locate and return an instance](#)
1249 [of a subclass of SCAClientFactory](#).

1250 8.10 [SCAClientFactoryFinder](#) Interface

1251 The [SCAClientFactoryFinder](#) interface is a [Service Provider Interface](#) representing a
1252 [SCAClientFactory](#) finder. [SCA](#) provides a default reference implementation of this interface. [SCA](#)
1253 [runtime vendors](#) can create alternative implementations of this interface that use different [class](#)
1254 [loading or lookup mechanisms](#).

1255 [package](#) [org.oasisopen.sca.client](#);

1256 [import](#) [java.util.Properties](#);

1257 [public interface](#) [SCAClientFactoryFinder](#) {

1258 [SCAClientFactory](#) [find](#)([Properties](#) [properties](#),
1259 [ClassLoader](#) [classLoader](#)
1260 [URI](#) [domainURI](#));
1261 }
1262
1263

Formatted: Bullets and Numbering

Formatted: Font: Not Bold, Not Italic, Complex Script
Font: Not Bold, Not Italic

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Indent: Before: 2.25", First line: 0.25"

Deleted: March

1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306

8.11 SCAClientFactoryFinderImpl Class

This class is a default implementation of an [SCAClientFactoryFinder](#), which is used to find an implementation of an [SCAClientFactory](#) subclass, as used to obtain an [SCAClient](#) object for use by a client. SCA runtime providers can replace this implementation with their own version.

```
package org.oasisopen.sca.client.impl;

public class SCAClientFactoryFinderImpl implements SCAClientFactoryFinder
{
    ...
    public SCAClientFactoryFinderImpl() {...}

    public SCAClientFactory find(Properties properties,
                               ClassLoader classLoader
                               URI domainURI)
        throws ServiceRuntimeException {...}
    ...
}
```

SCAClientFactoryFinderImpl () method:

Public constructor for the [SCAClientFactoryFinderImpl](#).

Returns:

- [SCAClientFactoryFinderImpl](#) which implements the [SCAClientFactoryFinder](#) interface

Parameters:

- *none*

Exceptions:

- *none*

find (Properties, ClassLoader, URI) method:

Obtains an implementation of the [SCAClientFactory](#) interface. It discovers a provider's [SCAClientFactory](#) implementation by referring to the following information in this order:

1. The [org.oasisopen.sca.client.SCAClientFactory](#) property from the [Properties](#) specified on the [newInstance\(\)](#) method call if specified
2. The [org.oasisopen.sca.client.SCAClientFactory](#) property from the [System Properties](#)
3. The [META-INF/services/org.oasisopen.sca.client.SCAClientFactory](#) file

Returns:

- [SCAClientFactory](#) implementation object

Parameters:

- *properties* - a set of [Properties](#) that can be used when creating the object which implements the [SCAClientFactory](#) interface.
- *classLoader* - a [ClassLoader](#) to use when creating the object which implements the [SCAClientFactory](#) interface.
- *domainURI* - a [URI](#) for the SCA Domain targeted by the [SCAClientFactory](#)

Exceptions:

- [ServiceRuntimeException](#) - if the [SCAClientFactory](#) implementation could not be found

Formatted: Indent: Before: 2.75", First line: 0.25"

Formatted: Indent: First line: 0.25"

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Font: Not Bold, Not Italic, Complex Script
Font: Not Bold, Not Italic

Formatted: Bulleted + Level: 1 + Aligned at: 0.5" + Tab after: 0.75" + Indent at: 0.75"

Formatted: Bullets and Numbering

Formatted: Font: Not Bold, Not Italic, Complex Script
Font: Not Bold, Not Italic

Deleted: March

1307 **8.12 NoSuchDomainException**

1308 The following shows the NoSuchDomainException:

1309 package org.oasisopen.sca;

1310 public class NoSuchDomainException extends Exception {
1311 ...
1312 }

1314 This exception indicates that the Domain specified could not be found.

1315 **8.13 NoSuchServiceException**

1316 The following shows the NoSuchServiceException:

1317 package org.oasisopen.sca;

1318 public class NoSuchServiceException extends Exception {
1319 ...
1320 }

1322 This exception indicates that the service specified could not be found.

1323 9 Java Annotations

1324 This section provides definitions of all the Java annotations which apply to SCA.

1325 This specification places constraints on some annotations that are not detectable by a Java
1326 compiler. For example, the definition of the @Property and @Reference annotations indicate that
1327 they are allowed on parameters, but sections 8.14 and 8.15 constrain those definitions to
1328 constructor parameters. An SCA runtime MUST verify the proper use of all SCA annotations and if
1329 an annotation is improperly used, the SCA runtime MUST NOT run the component which uses the
1330 invalid implementation code. [JCA90001]

1331 SCA annotations MUST NOT be used on static methods or on static fields. It is an error to use an
1332 SCA annotation on a static method or a static field of an implementation class and the SCA
1333 runtime MUST NOT instantiate such an implementation class. [JCA90002]

1334 9.1 @AllowsPassByReference

1335 The following Java code defines the **@AllowsPassByReference** annotation:

```
1336  
1337 package org.oasisopen.sca.annotation;  
1338  
1339 import static java.lang.annotation.ElementType.TYPE;  
1340 import static java.lang.annotation.ElementType.METHOD;  
1341 import static java.lang.annotation.RetentionPolicy.RUNTIME;  
1342 import java.lang.annotation.Retention;  
1343 import java.lang.annotation.Target;  
1344  
1345 @Target({TYPE, METHOD})  
1346 @Retention(RUNTIME)  
1347 public @interface AllowsPassByReference {  
1348  
1349 }  
1350
```

1351 The **@AllowsPassByReference** annotation is used on implementations of remotable interfaces to
1352 indicate that interactions with the service from a client within the same address space are allowed
1353 to use pass by reference data exchange semantics. The implementation promises that its by-value
1354 semantics will be maintained even if the parameters and return values are actually passed by-
1355 reference. This means that the service will not modify any operation input parameter or return
1356 value, even after returning from the operation. Either a whole class implementing a remotable
1357 service or an individual remotable service method implementation can be annotated using the
1358 @AllowsPassByReference annotation.

1359 @AllowsPassByReference has no attributes

1360 The following snippet shows a sample where @AllowsPassByReference is defined for the
1361 implementation of a service method on the Java component implementation class.

```
1362  
1363 @AllowsPassByReference  
1364 public String hello(String message) {  
1365     ...  
1366 }
```

1367 9.2 @Authentication

1368 The following Java code defines the **@Authentication** annotation:

```

1369
1370 package org.oasisopen.sca.annotation;
1371
1372 import static java.lang.annotation.ElementType.FIELD;
1373 import static java.lang.annotation.ElementType.METHOD;
1374 import static java.lang.annotation.ElementType.PARAMETER;
1375 import static java.lang.annotation.ElementType.TYPE;
1376 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1377 import static org.oasisopen.sca.Constants.SCA_PREFIX;
1378
1379 import java.lang.annotation.Inherited;
1380 import java.lang.annotation.Retention;
1381 import java.lang.annotation.Target;
1382
1383 @Inherited
1384 @Target({TYPE, FIELD, METHOD, PARAMETER})
1385 @Retention(RUNTIME)
1386 @Intent(Authentication.AUTHENTICATION)
1387 public @interface Authentication {
1388     String AUTHENTICATION = SCA_PREFIX + "authentication";
1389     String AUTHENTICATION_MESSAGE = AUTHENTICATION + ".message";
1390     String AUTHENTICATION_TRANSPORT = AUTHENTICATION + ".transport";
1391
1392     /**
1393      * List of authentication qualifiers (such as "message"
1394      * or "transport").
1395      *
1396      * @return authentication qualifiers
1397      */
1398     @Qualifier
1399     String[] value() default "";
1400 }

```

1401 The **@Authentication** annotation is used to indicate that the invocation requires authentication.
1402 See the [section on Application of Intent Annotations](#) for samples and details.

1403 9.3 @Callback

1404 The following Java code defines the **@Callback** annotation:

```

1405
1406 package org.oasisopen.sca.annotation;
1407
1408 import static java.lang.annotation.ElementType.TYPE;
1409 import static java.lang.annotation.ElementType.METHOD;
1410 import static java.lang.annotation.ElementType.FIELD;
1411 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1412 import java.lang.annotation.Retention;
1413 import java.lang.annotation.Target;
1414
1415 @Target(TYPE, METHOD, FIELD)
1416 @Retention(RUNTIME)
1417 public @interface Callback {
1418
1419     Class<?> value() default Void.class;
1420 }
1421
1422

```

Deleted: March

1423 The @Callback annotation is used to annotate a service interface with a callback interface by
1424 specifying the Java class object of the callback interface as an attribute.

1425 The @Callback annotation has the following attribute:

- 1426 • **value** – the name of a Java class file containing the callback interface

1427

1428 The @Callback annotation can also be used to annotate a method or a field of an SCA
1429 implementation class, in order to have a callback object injected. When used to annotate a
1430 method or a field of an implementation class for injection of a callback object, the @Callback
1431 annotation MUST NOT specify any attributes. [JCA90046]

1432 An example use of the @Callback annotation to declare a callback interface follows:

```
1433 package somepackage;  
1434 import org.oasisopen.sca.annotation.Callback;  
1435 import org.oasisopen.sca.annotation.Remotable;  
1436 @Remotable  
1437 @Callback(MyServiceCallback.class)  
1438 public interface MyService {  
1439  
1440     void someMethod(String arg);  
1441 }  
1442  
1443 @Remotable  
1444 public interface MyServiceCallback {  
1445  
1446     void receiveResult(String result);  
1447 }  
1448
```

1449 In this example, the implied component type is:

```
1450 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" >  
1451     <service name="MyService">  
1452         <interface.java interface="somepackage.MyService"  
1453             callbackInterface="somepackage.MyServiceCallback"/>  
1454     </service>  
1455 </componentType>
```

1457 9.4 @ComponentName

1458 The following Java code defines the @ComponentName annotation:

1459

```
1460 package org.oasisopen.sca.annotation;  
1461  
1462 import static java.lang.annotation.ElementType.METHOD;  
1463 import static java.lang.annotation.ElementType.FIELD;  
1464 import static java.lang.annotation.RetentionPolicy.RUNTIME;  
1465 import java.lang.annotation.Retention;  
1466 import java.lang.annotation.Target;  
1467  
1468 @Target({METHOD, FIELD})  
1469 @Retention(RUNTIME)  
1470 public @interface ComponentName {  
1471  
1472 }  
1473
```

1474 The @ComponentName annotation is used to denote a Java class field or setter method that is
1475 used to inject the component name.

1476 The following snippet shows a component name field definition sample.

1477

```
1478 @ComponentName  
1479 private String componentName;  
1480
```

1481 The following snippet shows a component name setter method sample.

1482

```
1483 @ComponentName  
1484 public void setComponentName(String name) {  
1485     //...  
1486 }
```

1487 9.5 @Confidentiality

1488 The following Java code defines the **@Confidentiality** annotation:

1489

```
1490 package org.oasisopen.sca.annotations;  
1491  
1492 import static java.lang.annotation.ElementType.FIELD;  
1493 import static java.lang.annotation.ElementType.METHOD;  
1494 import static java.lang.annotation.ElementType.PARAMETER;  
1495 import static java.lang.annotation.ElementType.TYPE;  
1496 import static java.lang.annotation.RetentionPolicy.RUNTIME;  
1497 import static org.oasisopen.sca.Constants.SCA_PREFIX;  
1498  
1499 import java.lang.annotation.Inherited;  
1500 import java.lang.annotation.Retention;  
1501 import java.lang.annotation.Target;  
1502  
1503 @Inherited  
1504 @Target({TYPE, FIELD, METHOD, PARAMETER})  
1505 @Retention(RUNTIME)  
1506 @Intent(Confidentiality.CONFIDENTIALITY)  
1507 public @interface Confidentiality {  
1508     String CONFIDENTIALITY = SCA_PREFIX + "confidentiality";  
1509     String CONFIDENTIALITY_MESSAGE = CONFIDENTIALITY + ".message";  
1510     String CONFIDENTIALITY_TRANSPORT = CONFIDENTIALITY + ".transport";  
1511  
1512     /**  
1513      * List of confidentiality qualifiers such as "message" or  
1514      * "transport".  
1515      *  
1516      * @return confidentiality qualifiers  
1517      */  
1518     @Qualifier  
1519     String[] value() default "";  
1520 }
```

1521 The **@Confidentiality** annotation is used to indicate that the invocation requires confidentiality.

1522 See the [section on Application of Intent Annotations](#) for samples and details.

1523 **9.6 @Constructor**

1524 The following Java code defines the **@Constructor** annotation:

```
1525 package org.oasisopen.sca.annotation;  
1526  
1527 import static java.lang.annotation.ElementType.CONSTRUCTOR;  
1528 import static java.lang.annotation.RetentionPolicy.RUNTIME;  
1529 import java.lang.annotation.Retention;  
1530 import java.lang.annotation.Target;  
1531  
1532 @Target({CONSTRUCTOR})  
1533 @Retention(RUNTIME)  
1534 public @interface Constructor {  
1535  
1536
```

1537 The @Constructor annotation is used to mark a particular constructor to use when instantiating a
1538 Java component implementation. If a constructor of an implementation class is annotated with
1539 @Constructor and the constructor has parameters, each of these parameters MUST have either a
1540 @Property annotation or a @Reference annotation. [JCA90003]

Comment [ME4]: There also needs to be a normative statement that at most 1 constructor can be annotated with @Constructor

1541 The following snippet shows a sample for the @Constructor annotation.

```
1542  
1543 public class HelloServiceImpl implements HelloService {  
1544     public HelloServiceImpl(){  
1545         ...  
1546     }  
1547  
1548     @Constructor  
1549     public HelloServiceImpl(@Property(name="someProperty")  
1550                             String someProperty ){  
1551         ...  
1552     }  
1553  
1554     public String hello(String message) {  
1555         ...  
1556     }  
1557 }  
1558
```

1559 **9.7 @Context**

1560 The following Java code defines the **@Context** annotation:

```
1561  
1562 package org.oasisopen.sca.annotation;  
1563  
1564 import static java.lang.annotation.ElementType.METHOD;  
1565 import static java.lang.annotation.ElementType.FIELD;  
1566 import static java.lang.annotation.RetentionPolicy.RUNTIME;  
1567 import java.lang.annotation.Retention;  
1568 import java.lang.annotation.Target;  
1569  
1570 @Target({METHOD, FIELD})  
1571 @Retention(RUNTIME)  
1572 public @interface Context {  
1573
```

Deleted: March

1574 }
1575

1576 The @Context annotation is used to denote a Java class field or a setter method that is used to
1577 inject a composite context for the component. The type of context to be injected is defined by the
1578 type of the Java class field or type of the setter method input argument; the type is either
1579 **ComponentContext** or **RequestContext**.

1580 The @Context annotation has no attributes.

1581 The following snippet shows a ComponentContext field definition sample.

```
1582  
1583 @Context  
1584 protected ComponentContext context;  
1585
```

1586 The following snippet shows a RequestContext field definition sample.

```
1587  
1588 @Context  
1589 protected RequestContext context;
```

1590 9.8 @Destroy

1591 The following Java code defines the **@Destroy** annotation:

```
1592  
1593 package org.oasisopen.sca.annotation;  
1594  
1595 import static java.lang.annotation.ElementType.METHOD;  
1596 import static java.lang.annotation.RetentionPolicy.RUNTIME;  
1597 import java.lang.annotation.Retention;  
1598 import java.lang.annotation.Target;  
1599  
1600 @Target(METHOD)  
1601 @Retention(RUNTIME)  
1602 public @interface Destroy {  
1603  
1604 }  
1605
```

1606 The @Destroy annotation is used to denote a single Java class method that will be called when the
1607 scope defined for the implementation class ends. A method annotated with @Destroy MAY have
1608 any access modifier and MUST have a void return type and no arguments. [JCA90004]

1609 If there is a method annotated with @Destroy that matches the criteria for the annotation, the
1610 SCA runtime MUST call the annotated method when the scope defined for the implementation
1611 class ends. [JCA90005]

1612 The following snippet shows a sample for a destroy method definition.

```
1613  
1614 @Destroy  
1615 public void myDestroyMethod() {  
1616     ...  
1617 }
```

1618 9.9 @EagerInit

1619 The following Java code defines the **@EagerInit** annotation:

1620

```
1621 package org.oasisopen.sca.annotation;
1622
1623 import static java.lang.annotation.ElementType.TYPE;
1624 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1625 import java.lang.annotation.Retention;
1626 import java.lang.annotation.Target;
1627
1628 @Target(TYPE)
1629 @Retention(RUNTIME)
1630 public @interface EagerInit {
```

1631

1632

1633

1634 The **@EagerInit** annotation is used to mark the Java class of a COMPOSITE scoped
1635 implementation for eager initialization. When marked for eager initialization with an @EagerInit
1636 annotation, the composite scoped instance MUST be created when its containing component is
1637 started. [JCA90007]

1638 9.10 @Init

1639 The following Java code defines the **@Init** annotation:

1640

```
1641 package org.oasisopen.sca.annotation;
1642
1643 import static java.lang.annotation.ElementType.METHOD;
1644 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1645 import java.lang.annotation.Retention;
1646 import java.lang.annotation.Target;
1647
1648 @Target(METHOD)
1649 @Retention(RUNTIME)
1650 public @interface Init {
```

1651

1652

1653

1654

1655 The @Init annotation is used to denote a single Java class method that is called when the scope
1656 defined for the implementation class starts. A method marked with the @Init annotation MAY
1657 have any access modifier and MUST have a void return type and no arguments. [JCA90008]

1658 If there is a method annotated with @Init that matches the criteria for the annotation, the SCA
1659 runtime MUST call the annotated method after all property and reference injection is complete.
1660 [JCA90009]

1661 The following snippet shows an example of an init method definition.

1662

```
1663 @Init
1664 public void myInitMethod() {
1665     ...
1666 }
```

1667 9.11 @Integrity

1668 The following Java code defines the **@Integrity** annotation:

1669

Deleted: If there is a method annotated with @Init that matches the criteria for the annotation, the SCA runtime MUST call the annotated method after all property and reference injection is complete.

Deleted: If there is a method annotated with @Init that matches the criteria for the annotation, the SCA runtime MUST call the annotated method after all property and reference injection is complete.

Deleted: March

```

1670 package org.oasisopen.sca.annotation;
1671
1672 import static java.lang.annotation.ElementType.FIELD;
1673 import static java.lang.annotation.ElementType.METHOD;
1674 import static java.lang.annotation.ElementType.PARAMETER;
1675 import static java.lang.annotation.ElementType.TYPE;
1676 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1677 import static org.oasisopen.Constants.SCA_PREFIX;
1678
1679 import java.lang.annotation.Inherited;
1680 import java.lang.annotation.Retention;
1681 import java.lang.annotation.Target;
1682
1683 @Inherited
1684 @Target({TYPE, FIELD, METHOD, PARAMETER})
1685 @Retention(RUNTIME)
1686 @Intent(Integrity.INTEGRITY)
1687 public @interface Integrity {
1688     String INTEGRITY = SCA_PREFIX + "integrity";
1689     String INTEGRITY_MESSAGE = INTEGRITY + ".message";
1690     String INTEGRITY_TRANSPORT = INTEGRITY + ".transport";
1691
1692     /**
1693      * List of integrity qualifiers (such as "message" or "transport").
1694      *
1695      * @return integrity qualifiers
1696      */
1697     @Qualifier
1698     String[] value() default "";
1699 }

```

1701 The **@Integrity** annotation is used to indicate that the invocation requires integrity (ie no
1702 tampering of the messages between client and service).

1703 See the [section on Application of Intent Annotations](#) for samples and details.

1704 9.12 @Intent

1705 The following Java code defines the **@Intent** annotation:

```

1706 package org.oasisopen.sca.annotation;
1707
1708 import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
1709 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1710 import java.lang.annotation.Retention;
1711 import java.lang.annotation.Target;
1712
1713 @Target({ANNOTATION_TYPE})
1714 @Retention(RUNTIME)
1715 public @interface Intent {
1716     /**
1717      * The qualified name of the intent, in the form defined by
1718      * {@link javax.xml.namespace.QName#toString}.
1719      * @return the qualified name of the intent
1720      */
1721     String value() default "";
1722
1723     /**
1724

```

Deleted: March

```

1725     * The XML namespace for the intent.
1726     * @return the XML namespace for the intent
1727     */
1728     String targetNamespace() default "";
1729
1730     /**
1731     * The name of the intent within its namespace.
1732     * @return name of the intent within its namespace
1733     */
1734     String localPart() default "";
1735 }
1736

```

1737 The @Intent annotation is used for the creation of new annotations for specific intents. It is not
 1738 expected that the @Intent annotation will be used in application code.

1739 See the [section "How to Create Specific Intent Annotations"](#) for details and samples of how to
 1740 define new intent annotations.

1741 9.13 @OneWay

1742 The following Java code defines the **@OneWay** annotation:

```

1743
1744 package org.oasisopen.sca.annotation;
1745
1746 import static java.lang.annotation.ElementType.METHOD;
1747 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1748 import java.lang.annotation.Retention;
1749 import java.lang.annotation.Target;
1750
1751 @Target(METHOD)
1752 @Retention(RUNTIME)
1753 public @interface OneWay {
1754
1755 }
1756
1757

```

1758 The @OneWay annotation is used on a Java interface or class method to indicate that invocations
 1759 will be dispatched in a non-blocking fashion as described in the section on Asynchronous
 1760 [Programming](#).

Comment [ME5]: Needs recasting in a normative form of statement

1761 The @OneWay annotation has no attributes.

1762 The following snippet shows the use of the @OneWay annotation on an interface.

```

1763 package services.hello;
1764
1765 import org.oasisopen.sca.annotation.OneWay;
1766
1767 public interface HelloService {
1768     @OneWay
1769     void hello(String name);
1770 }

```

1771 9.14 @PolicySets

1772 The following Java code defines the **@PolicySets** annotation:

```

1773 package org.oasisopen.sca.annotation;
1774

```

Deleted: March

```

1775
1776 import static java.lang.annotation.ElementType.FIELD;
1777 import static java.lang.annotation.ElementType.METHOD;
1778 import static java.lang.annotation.ElementType.PARAMETER;
1779 import static java.lang.annotation.ElementType.TYPE;
1780 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1781
1782 import java.lang.annotation.Retention;
1783 import java.lang.annotation.Target;
1784
1785 @Target({TYPE, FIELD, METHOD, PARAMETER})
1786 @Retention(RUNTIME)
1787 public @interface PolicySets {
1788     /**
1789      * Returns the policy sets to be applied.
1790      *
1791      * @return the policy sets to be applied
1792      */
1793     String[] value() default "";
1794 }
1795

```

1796 The **@PolicySets** annotation is used to attach one or more SCA Policy Sets to a Java implementation class or to one of its subelements.

1797 See the [section "Policy Set Annotations"](#) for details and samples.

1799 9.15 @Property

1800 The following Java code defines the **@Property** annotation:

```

1801 package org.oasisopen.sca.annotation;
1802
1803 import static java.lang.annotation.ElementType.METHOD;
1804 import static java.lang.annotation.ElementType.FIELD;
1805 import static java.lang.annotation.ElementType.PARAMETER;
1806 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1807 import java.lang.annotation.Retention;
1808 import java.lang.annotation.Target;
1809
1810 @Target({METHOD, FIELD, PARAMETER})
1811 @Retention(RUNTIME)
1812 public @interface Property {
1813
1814     String name() default "";
1815     boolean required() default true;
1816 }
1817

```

1818 The @Property annotation is used to denote a Java class field, a setter method, or a constructor parameter that is used to inject an SCA property value. The type of the property injected, which can be a simple Java type or a complex Java type, is defined by the type of the Java class field or the type of the input parameter of the setter method or constructor.

1822 The @Property annotation can be used on fields, on setter methods or on a constructor method parameter. However, **the @Property annotation MUST NOT be used on a class field that is declared as final.** [JCA90011]

1825 Properties can also be injected via setter methods even when the @Property annotation is not present. However, **the @Property annotation MUST be used in order to inject a property onto a non-public field.** [JCA90012] In the case where there is no @Property annotation, the name of the property is the same as the name of the field or setter.

Deleted: the @Property annotation MUST NOT be used on a class field that is declared as final.

Deleted: the @Property annotation MUST NOT be used on a class field that is declared as final.

Deleted: March

1829 Where there is both a setter method and a field for a property, the setter method is used.

1830 The @Property annotation has the following attributes:

- 1831 • **name (optional)** – the name of the property. For a field annotation, the default is the
1832 name of the field of the Java class. For a setter method annotation, the default is the
1833 JavaBeans property name [JAVABEANS] corresponding to the setter method name. For a
1834 @Property annotation applied to a constructor parameter, there is no default value for the
1835 name attribute and the name attribute MUST be present. [JCA90013]
- 1836 • **required (optional)** – a boolean value which specifies whether injection of the property
1837 value is required or not, where true means injection is required and false means injection
1838 is not required. Defaults to true. For a @Property annotation applied to a constructor
1839 parameter, the required attribute MUST have the value true. [JCA90014]

1840

1841 The following snippet shows a property field definition sample.

1842

```
1843 @Property(name="currency", required=true)  
1844 protected String currency;
```

1845

1846 The following snippet shows a property setter sample

1847

```
1848 @Property(name="currency", required=true)  
1849 public void setCurrency( String theCurrency ) {  
1850     ....  
1851 }
```

1852

1853 For a @Property annotation, if the the type of the Java class field or the type of the input
1854 parameter of the setter method or constructor is defined as an array or as any type that extends
1855 or implements java.util.Collection, then the SCA runtime MUST introspect the component type of
1856 the implementation with a <property/> element with a @many attribute set to true, otherwise
1857 @many MUST be set to false. [JCA90047]

1858 The following snippet shows the definition of a configuration property using the @Property
1859 annotation for a collection.

```
1860 ...  
1861 private List<String> helloConfigurationProperty;  
1862  
1863 @Property(required=true)  
1864 public void setHelloConfigurationProperty(List<String> property) {  
1865     helloConfigurationProperty = property;  
1866 }  
1867 ...
```

1868 9.16 @Qualifier

1869 The following Java code defines the @Qualifier annotation:

```
1870 package org.oasisopen.sca.annotation;  
1871  
1872 import static java.lang.annotation.ElementType.METHOD;  
1873 import static java.lang.annotation.RetentionPolicy.RUNTIME;
```

1875

Deleted: For a @Property annotation, if the the type of the Java class field or the type of the input parameter of the setter method or constructor is defined as an array or as any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <property/> element with a @many attribute set to true, otherwise @many MUST be set to false.

Deleted: For a @Property annotation, if the the type of the Java class field or the type of the input parameter of the setter method or constructor is defined as an array or as any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <property/> element with a @many attribute set to true, otherwise @many MUST be set to false.

Deleted: March

```

1876 import java.lang.annotation.Retention;
1877 import java.lang.annotation.Target;
1878
1879 @Target(METHOD)
1880 @Retention(RUNTIME)
1881 public @interface Qualifier {
1882 }
1883

```

1884 The @Qualifier annotation is applied to an attribute of a specific intent annotation definition,
 1885 defined using the @Intent annotation, to indicate that the attribute provides qualifiers for the
 1886 intent. The @Qualifier annotation MUST be used in a specific intent annotation definition where the
 1887 intent has qualifiers. [JCA90015]

1888 See the [section "How to Create Specific Intent Annotations"](#) for details and samples of how to
 1889 define new intent annotations.

1890 9.17 @Reference

1891 The following Java code defines the **@Reference** annotation:

```

1892
1893 package org.oasisopen.sca.annotation;
1894
1895 import static java.lang.annotation.ElementType.METHOD;
1896 import static java.lang.annotation.ElementType.FIELD;
1897 import static java.lang.annotation.ElementType.PARAMETER;
1898 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1899 import java.lang.annotation.Retention;
1900 import java.lang.annotation.Target;
1901 @Target({METHOD, FIELD, PARAMETER})
1902 @Retention(RUNTIME)
1903 public @interface Reference {
1904
1905     String name() default "";
1906     boolean required() default true;
1907 }
1908

```

1909 The @Reference annotation type is used to annotate a Java class field, a setter method, or a
 1910 constructor parameter that is used to inject a service that resolves the reference. The interface of
 1911 the service injected is defined by the type of the Java class field or the type of the input parameter
 1912 of the setter method or constructor.

1913 The @Reference annotation MUST NOT be used on a class field that is declared as final.
 1914 [JCA90016]

1915 References can also be injected via setter methods even when the @Reference annotation is not
 1916 present. However, the @Reference annotation MUST be used in order to inject a reference onto a
 1917 non-public field. [JCA90017] In the case where there is no @Reference annotation, the name of
 1918 the reference is the same as the name of the field or setter.

1919 Where there is both a setter method and a field for a reference, the setter method is used.

1920 The @Reference annotation has the following attributes:

- 1921 • **name : String (optional)** – the name of the reference. For a field annotation, the default is
 1922 the name of the field of the Java class. For a setter method annotation, the default is the
 1923 JavaBeans property name corresponding to the setter method name. For a @Reference
 1924 annotation applied to a constructor parameter, there is no default for the name attribute
 1925 and the name attribute MUST be present. [JCA90018]

- 1926
- **required (optional)** – a boolean value which specifies whether injection of the service reference is required or not, where true means injection is required and false means injection is not required. Defaults to true. For a @Reference annotation applied to a constructor parameter, the required attribute MUST have the value true. [JCA90019]
- 1927
- 1928
- 1929

1930

1931 The following snippet shows a reference field definition sample.

1932

```
1933 @Reference(name="stockQuote", required=true)
1934 protected StockQuoteService stockQuote;
```

1935

1936 The following snippet shows a reference setter sample

1937

```
1938 @Reference(name="stockQuote", required=true)
1939 public void setStockQuote( StockQuoteService theSQService ) {
1940     ...
1941 }
```

1942

1943 The following fragment from a component implementation shows a sample of a service reference using the @Reference annotation. The name of the reference is "helloService" and its type is HelloService. The clientMethod() calls the "hello" operation of the service referenced by the helloService reference.

1947

```
1948 package services.hello;
1949
1950 private HelloService helloService;
1951
1952 @Reference(name="helloService", required=true)
1953 public setHelloService(HelloService service) {
1954     helloService = service;
1955 }
1956
1957 public void clientMethod() {
1958     String result = helloService.hello("Hello World!");
1959     ...
1960 }
1961
```

1962 The presence of a @Reference annotation is reflected in the componentType information that the runtime generates through reflection on the implementation class. The following snippet shows the component type for the above component implementation fragment.

1965

```
1966 <?xml version="1.0" encoding="ASCII"?>
1967 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712">
1968     <!-- Any services offered by the component would be listed here -->
1969     <reference name="helloService" multiplicity="1..1">
1970         <interface.java interface="services.hello.HelloService"/>
1971     </reference>
1972 </componentType>
```

1973

1974

1975

Deleted: March

1976 If the type of a reference is not an array or any type that extends or implements
1977 java.util.Collection, then the SCA runtime MUST introspect the component type of the
1978 implementation with a <reference/> element with @multiplicity= 0..1 if the @Reference
1979 annotation required attribute is false and with @multiplicity=1..1 if the @Reference annotation
1980 required attribute is true. [JCA90020]

1981 If the type of a reference is defined as an array or as any type that extends or implements
1982 java.util.Collection, then the SCA runtime MUST introspect the component type of the
1983 implementation with a <reference/> element with @multiplicity=0..n if the @Reference
1984 annotation required attribute is false and with @multiplicity=1..n if the @Reference annotation
1985 required attribute is true. [JCA90021]

1986 The following fragment from a component implementation shows a sample of a service reference
1987 definition using the @Reference annotation on a java.util.List. The name of the reference is
1988 "helloServices" and its type is HelloService. The clientMethod() calls the "hello" operation of all the
1989 services referenced by the helloServices reference. In this case, at least one HelloService needs
1990 to be present, so **required** is true.

```
1991 @Reference(name="helloServices", required=true)  
1992 protected List<HelloService> helloServices;  
1993  
1994 public void clientMethod() {  
1995     ...  
1996     for (int index = 0; index < helloServices.size(); index++) {  
1997         HelloService helloService =  
1998             (HelloService)helloServices.get(index);  
1999         String result = helloService.hello("Hello World!");  
2000     }  
2001     ...  
2002 }  
2003 }  
2004 }  
2005 }
```

2006 The following snippet shows the XML representation of the component type reflected from for the
2007 former component implementation fragment. There is no need to author this component type in
2008 this case since it can be reflected from the Java class.

```
2009  
2010 <?xml version="1.0" encoding="ASCII"?>  
2011 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712">  
2012     <!-- Any services offered by the component would be listed here -->  
2013     <reference name="helloServices" multiplicity="1..n">  
2014         <interface.java interface="services.hello.HelloService"/>  
2015     </reference>  
2016 </componentType>  
2017  
2018
```

2019
2020 An unwired reference with a multiplicity of 0..1 MUST be presented to the implementation code by
2021 the SCA runtime as null. [JCA90022] An unwired reference with a multiplicity of 0..n MUST be
2022 presented to the implementation code by the SCA runtime as an empty array or empty collection.
2023 [JCA90023]

2024 9.17.1 Reinjection

2025 References MAY be reinjected by an SCA runtime after the initial creation of a component if the
2026 reference target changes due to a change in wiring that has occurred since the component was
2027 initialized. [JCA90024]

2028 In order for reinjection to occur, the following MUST be true:

Deleted: If the type of a reference is not an array or any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference/> element with @multiplicity= 0..1 if the @Reference annotation required attribute is false and with @multiplicity=1..1 if the @Reference annotation required attribute is true.

Deleted: If the type of a reference is not an array or any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference/> ... [1]

Deleted: If the type of a reference is defined as an array or as any type that extends or implements java.util.Collection, then the SCA runtime MUST ... [2]

Deleted: If the type of a reference is defined as an array or as any type that extends or implements java.util.Collection, then the SCA runtime MUST ... [3]

Deleted: An unwired reference with a multiplicity of 0..1 MUST be presented to the implementation code by the SCA runtime as null

Deleted: An unwired reference with a multiplicity of 0..1 MUST be presented to the implementation code by the SCA runtime as null

Deleted: An unwired reference with a multiplicity of 0..n MUST be presented to the implementation code by the SCA runtime as a ... [4]

Deleted: An unwired reference with a multiplicity of 0..n MUST be presented to the implementation code by the SCA runtime as a ... [5]

Deleted: March

2029 1. The component MUST NOT be STATELESS scoped.

2030 2. The reference MUST use either field-based injection or setter injection. References that are

2031 injected through constructor injection MUST NOT be changed.

2032 [JCA90025]

2033 Setter injection allows for code in the setter method to perform processing in reaction to a change.

2034 If a reference target changes and the reference is not reinjected, the reference MUST continue to

2035 work as if the reference target was not changed. [JCA90026]

2036 If an operation is called on a reference where the target of that reference has been undeployed,

2037 the SCA runtime SHOULD throw an InvalidServiceException. [JCA90027] If an operation is called

2038 on a reference where the target of the reference has become unavailable for some reason, the

2039 SCA runtime SHOULD throw a ServiceUnavailableException. [JCA90028] If the target service of

2040 the reference is changed, the reference MAY continue to work, depending on the runtime and the

2041 type of change that was made. [JCA90029] If it doesn't work, the exception thrown will depend on

2042 the runtime and the cause of the failure.

2043 A ServiceReference that has been obtained from a reference by ComponentContext.cast()

2044 corresponds to the reference that is passed as a parameter to cast(). If the reference is

2045 subsequently reinjected, the ServiceReference obtained from the original reference MUST continue

2046 to work as if the reference target was not changed. [JCA90030] If the target of a ServiceReference

2047 has been undeployed, the SCA runtime SHOULD throw a InvalidServiceException when an

2048 operation is invoked on the ServiceReference. [JCA90031] If the target of a ServiceReference has

2049 become unavailable, the SCA runtime SHOULD throw a ServiceUnavailableException when an

2050 operation is invoked on the ServiceReference. [JCA90032] If the target service of a

2051 ServiceReference is changed, the reference MAY continue to work, depending on the runtime and

2052 the type of change that was made. [JCA90033] If it doesn't work, the exception thrown will

2053 depend on the runtime and the cause of the failure.

2054 A reference or ServiceReference accessed through the component context by calling getService()

2055 or getServiceReference() MUST correspond to the current configuration of the domain. This applies

2056 whether or not reinjection has taken place. [JCA90034] If the target of a reference or

2057 ServiceReference accessed through the component context by calling getService() or

2058 getServiceReference() has been undeployed or has become unavailable, the result SHOULD be a

2059 reference to the undeployed or unavailable service, and attempts to call business methods

2060 SHOULD throw an InvalidServiceException or a ServiceUnavailableException. [JCA90035] If the

2061 target service of a reference or ServiceReference accessed through the component context by

2062 calling getService() or getServiceReference() has changed, the returned value SHOULD be a

2063 reference to the changed service. [JCA90036]

2064 The rules for reference reinjection also apply to references with a multiplicity of 0..n or 1..n. This

2065 means that in the cases where reference reinjection is not allowed, the array or Collection for a

2066 reference of multiplicity 0..n or multiplicity 1..n MUST NOT change its contents when changes

2067 occur to the reference wiring or to the targets of the wiring. [JCA90037] In cases where the

2068 contents of a reference array or collection change when the wiring changes or the targets change,

2069 then for references that use setter injection, the setter method MUST be called by the SCA

2070 runtime for any change to the contents. [JCA90038] A reinjected array or Collection for a

2071 reference MUST NOT be the same array or Collection object previously injected to the component.

2072 [JCA90039]

2073

Deleted: In order for reinjection to occur, the following MUST be true:
 1. The component MUST NOT be STATELESS scoped.
 2. The reference MUST use either field-based injection or setter injection. References that are injected through constructor injection MUST NOT be changed.

Deleted: In order for reinjection to occur, the following MUST be true:
 1. The component MUST NOT be STATELESS scoped.
 2. The reference MUST use either field-based injection or setter injection. References that are injected through constructor injection MUST NOT be changed.

Deleted: If the target service of the reference is changed, the referen ... [6]

Deleted: If the target service of the reference is changed, the referen ... [7]

Deleted: If the target service of a ServiceReference is ... [8]

Deleted: If the target service of a ServiceReference is ... [9]

Deleted: A reference or ServiceReference accessed through the compon ... [10]

Deleted: A reference or ServiceReference accessed through the compon ... [11]

Deleted: If the target of a reference or ServiceReference ac ... [12]

Deleted: If the target of a reference or ServiceReference ac ... [13]

Deleted: If the target service of a reference or ServiceReference ac ... [14]

Deleted: If the target service of a reference or ServiceReference ac ... [15]

Deleted: In cases where the contents of a reference array or collection c ... [16]

Deleted: In cases where the contents of a reference array or collection c ... [17]

Deleted: March

Change event	Effect on		
	Injected Reference or ServiceReference	Existing ServiceReference Object**	Subsequent invocations of ComponentContext.getServiceReference() or getService()
Change to the target of the reference	can be reinjected (if other conditions* apply). If not reinjected, then it	continue to work as if the reference target was not changed.	Result corresponds to the current configuration of the domain.

	continues to work as if the reference target was not changed.		
Target service undeployed	Business methods throw InvalidServiceException.	Business methods throw InvalidServiceException.	Result is a reference to the undeployed service. Business methods throw InvalidServiceException.
Target service becomes unavailable	Business methods throw ServiceUnavailableException	Business methods throw ServiceUnavailableException	Result is be a reference to the unavailable service. Business methods throw ServiceUnavailableException.
Target service changed	might continue to work, depending on the runtime and the type of change that was made. If it doesn't work, the exception thrown will depend on the runtime and the cause of the failure.	might continue to work, depending on the runtime and the type of change that was made. If it doesn't work, the exception thrown will depend on the runtime and the cause of the failure.	Result is a reference to the changed service.
<p>* Other conditions:</p> <p>The component cannot be STATELESS scoped.</p> <p>The reference has to use either field-based injection or setter injection. References that are injected through constructor injection cannot be changed.</p> <p>** Result of invoking ComponentContext.cast() corresponds to the reference that is passed as a parameter to cast().</p>			

2074

2075 9.18 @Remotable

2076 The following Java code defines the **@Remotable** annotation:

2077

```
2078 package org.oasisopen.sca.annotation;
2079
2080 import static java.lang.annotation.ElementType.TYPE;
2081 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2082 import java.lang.annotation.Retention;
2083 import java.lang.annotation.Target;
```

2084

```
2085
2086 @Target(TYPE)
2087 @Retention(RUNTIME)
2088 public @interface Remotable {
2089
2090 }
2091
```

2092 The @Remotable annotation is used to specify a Java service interface as remotable. A remotable
 2093 service can be published externally as a service and MUST be translatable into a WSDL portType.
 2094 [JCA90040]

2095 The @Remotable annotation has no attributes.

2096 The following snippet shows the Java interface for a remotable service with its @Remotable
 2097 annotation.

Deleted: March

```

2098 package services.hello;
2099
2100 import org.oasisopen.sca.annotation.*;
2101
2102 @Remotable
2103 public interface HelloService {
2104     String hello(String message);
2105 }
2106
2107

```

2108 The style of remotable interfaces is typically **coarse grained** and intended for **loosely coupled**
2109 interactions. Remotable service interfaces are not allowed to make use of method **overloading**.

2110 Complex data types exchanged via remotable service interfaces need to be compatible with the
2111 marshalling technology used by the service binding. For example, if the service is going to be
2112 exposed using the standard Web Service binding, then the parameters can be JAXB [JAX-B] types
2113 or they can be Service Data Objects (SDOs) [SDO].

2114 Independent of whether the remotable service is called from outside of the composite that
2115 contains it or from another component in the same composite, the data exchange semantics are
2116 **by-value**.

2117 Implementations of remotable services can modify input data during or after an invocation and
2118 can modify return data after the invocation. If a remotable service is called locally or remotely, the
2119 SCA container is responsible for making sure that no modification of input data or post-invocation
2120 modifications to return data are seen by the caller.

2121 The following snippet shows a remotable Java service interface.

```

2122
2123 package services.hello;
2124
2125 import org.oasisopen.sca.annotation.*;
2126
2127 @Remotable
2128 public interface HelloService {
2129     String hello(String message);
2130 }
2131
2132 package services.hello;
2133
2134 import org.oasisopen.sca.annotation.*;
2135
2136 @Service(HelloService.class)
2137 public class HelloServiceImpl implements HelloService {
2138     public String hello(String message) {
2139         ...
2140     }
2141 }
2142
2143

```

2144 9.19 @Requires

2145 The following Java code defines the **@Requires** annotation:

```

2146 package org.oasisopen.sca.annotation;
2147
2148 import static java.lang.annotation.ElementType.FIELD;
2149

```

Deleted: March


```

2150 import static java.lang.annotation.ElementType.METHOD;
2151 import static java.lang.annotation.ElementType.PARAMETER;
2152 import static java.lang.annotation.ElementType.TYPE;
2153 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2154
2155 import java.lang.annotation.Inherited;
2156 import java.lang.annotation.Retention;
2157 import java.lang.annotation.Target;
2158
2159 @Inherited
2160 @Retention(RUNTIME)
2161 @Target({TYPE, METHOD, FIELD, PARAMETER})
2162 public @interface Requires {
2163     /**
2164      * Returns the attached intents.
2165      *
2166      * @return the attached intents
2167      */
2168     String[] value() default "";
2169 }

```

2171 The **@Requires** annotation supports general purpose intents specified as strings. Users can also
2172 define specific intent annotations using the @Intent annotation.

2173 See the [section "General Intent Annotations"](#) for details and samples.

2174 9.20 @Scope

2175 The following Java code defines the **@Scope** annotation:

```

2176 package org.oasisopen.sca.annotation;
2177
2178 import static java.lang.annotation.ElementType.TYPE;
2179 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2180 import java.lang.annotation.Retention;
2181 import java.lang.annotation.Target;
2182
2183 @Target(TYPE)
2184 @Retention(RUNTIME)
2185 public @interface Scope {
2186
2187     String value() default "STATELESS";
2188 }

```

2189 The @Scope annotation MUST only be used on a service's implementation class. It is an error to
2190 use this annotation on an interface. [JCA90041]

2191 The @Scope annotation has the following attribute:

- 2192 • **value** – the name of the scope.
2193 SCA defines the following scope names, but others can be defined by particular Java-
2194 based implementation types:
2195 STATELESS
2196 COMPOSITE
2197 For 'STATELESS' implementations, a different implementation instance can be used to
2198 service each request. Implementation instances can be newly created or be drawn from a
2199 pool of instances.

2200 The default value is STATELESS.

2201 The following snippet shows a sample for a COMPOSITE scoped service implementation:

```

2202 package services.hello;

```

Deleted: March


```

2203
2204 import org.oasisopen.sca.annotation.*;
2205
2206 @Service(HelloService.class)
2207 @Scope("COMPOSITE")
2208 public class HelloServiceImpl implements HelloService {
2209
2210     public String hello(String message) {
2211         ...
2212     }
2213 }
2214

```

9.21 @Service

The following Java code defines the **@Service** annotation:

```

2217 package org.oasisopen.sca.annotation;
2218
2219 import static java.lang.annotation.ElementType.TYPE;
2220 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2221 import java.lang.annotation.Retention;
2222 import java.lang.annotation.Target;
2223
2224 @Target(TYPE)
2225 @Retention(RUNTIME)
2226 public @interface Service {
2227
2228     Class<?>[] interfaces() default {};
2229     Class<?> value() default Void.class;
2230 }
2231

```

The @Service annotation is used on a component implementation class to specify the SCA services offered by the implementation. An implementation class need not be declared as implementing all of the interfaces implied by the services declared in its @Service annotation, but all methods of all the declared service interfaces MUST be present. [JCA90042] A class used as the implementation of a service is not required to have a @Service annotation. If a class has no @Service annotation, then the rules determining which services are offered and what interfaces those services have are determined by the specific implementation type.

The @Service annotation has the following attributes:

- **interfaces (1..1)** – The value is an array of interface or class objects that are exposed as services by this component implementation.
- **value** – A shortcut for the case when the class provides only a single service interface - contains a single interface or class object that is exposed as a service by this component implementation.

A @Service annotation MUST only have one of the interfaces attribute or value attribute specified. [JCA90043]

A @Service annotation with no attributes MUST be ignored, it is the same as not having the annotation there at all. [JCA90044]

The **service names** of the defined services default to the names of the interfaces or class, without the package name.

Formatted: Pattern: Clear (Light Yellow)

Deleted: An implementation class need not be declared as implementing all of the interfaces implied by the services declared in its @Service annotation, but all methods of all the declared service interfaces MUST be present.

Deleted: An implementation class need not be declared as implementing all of the interfaces implied by the services declared in its @Service annotation, but all methods of all the declared service interfaces MUST be present.

Deleted: March

2252 **A component implementation MUST NOT have two services with the same Java simple name.**
2253 **[JCA90045]** If a Java implementation needs to realize two services with the same Java simple
2254 name then this can be achieved through subclassing of the interface.

2255 The following snippet shows an implementation of the HelloService marked with the @Service
2256 annotation.

```
2257 package services.hello;  
2258  
2259 import org.oasisopen.sca.annotation.Service;  
2260  
2261 @Service(HelloService.class)  
2262 public class HelloServiceImpl implements HelloService {  
2263     public void hello(String name) {  
2264         System.out.println("Hello " + name);  
2265     }  
2266 }  
2267  
2268
```

2269

10 WSDL to Java and Java to WSDL

2270
2271
2272

The SCA Client and Implementation Model for Java applies the WSDL to Java and Java to WSDL mapping rules as defined by the JAX-WS specification [JAX-WS] for generating remotable Java interfaces from WSDL portTypes and vice versa.

2273
2274
2275
2276
2277
2278

For the purposes of the Java-to-WSDL mapping algorithm, the SCA runtime MUST treat a Java interface as if it had a @WebService annotation on the class, even if it doesn't. [JCA100001] The SCA runtime MUST treat an @org.oasisopen.sca.annotation.OneWay annotation as a synonym for the @javax.jws.OneWay annotation. [JCA100002] For the WSDL-to-Java mapping, the SCA runtime MUST take the generated @WebService annotation to imply that the Java interface is @Remotable. [JCA100003]

2279
2280
2281
2282
2283
2284

For the mapping from Java types to XML schema types, SCA permits both the JAXB 2.1 [JAX-B] mapping and the SDO 2.1 [SDO] mapping. SCA runtimes MUST support the JAXB 2.1 mapping from Java types to XML schema types. [JCA100004] SCA runtimes MAY support the SDO 2.1 mapping from Java types to XML schema types. [JCA100005] Having a choice of binding technologies is allowed, as noted in the first paragraph of section 5 of the JSR 181 (version 2) specification, which is referenced by the JAX-WS specification.

2285
2286

The JAX-WS mappings are applied with the following restrictions:

- No support for holders

2287

2288
2289

Note: This specification needs more examples and discussion of how JAX-WS's client asynchronous model is used.

2290

10.1 JAX-WS Client Asynchronous API for a Synchronous Service

2291
2292
2293
2294
2295

The JAX-WS specification defines a mapping of a synchronous service invocation, which provides a client application with a means of invoking that service asynchronously, so that the client can invoke a service operation and proceed to do other work without waiting for the service operation to complete its processing. The client application can retrieve the results of the service either through a polling mechanism or via a callback method which is invoked when the operation completes.

2296
2297
2298
2299
2300
2301
2302
2303

For SCA service interfaces defined using interface.java, the Java interface MUST NOT contain the additional client-side asynchronous polling and callback methods defined by JAX-WS. [JCA100006] For SCA reference interfaces defined using interface.java, the Java interface MAY contain the additional client-side asynchronous polling and callback methods defined by JAX-WS. [JCA100007] If the additional client-side asynchronous polling and callback methods defined by JAX-WS are present in the interface which declares the type of a reference in the implementation, SCA Runtimes MUST NOT include these methods in the SCA reference interface in the component type of the implementation. [JCA100008]

2304
2305

The additional client-side asynchronous polling and callback methods defined by JAX-WS are recognized in a Java interface as follows:

2306
2307
2308
2309

- For each method M in the interface, if another method P in the interface has
- a method name that is M's method name with the characters "Async" appended, and
 - the same parameter signature as M, and
 - a return type of Response<R> where R is the return type of M

2310

then P is a JAX-WS polling method that isn't part of the SCA interface contract.

2311

For each method M in the interface, if another method C in the interface has

2312
2313
2314

- a method name that is M's method name with the characters "Async" appended, and
- a parameter signature that is M's parameter signature with an additional final parameter of type AsyncHandler<R> where R is the return type of M, and

Deleted: For the purposes of the Java-to-WSDL mapping algorithm, the SCA runtime MUST treat a Java interface as if it had a @WebService annotation on the class, even if it doesn't.

Deleted: For the purposes of the Java-to-WSDL mapping algorithm, the SCA runtime MUST treat a Java interface as if it had a @WebService annotation on the class, even if it doesn't.

Deleted: The SCA runtime MUST treat an @org.oasisopen.sca.annotation.OneWay annotation as a synonym for the @javax.jws.OneWay annotation.

Deleted: The SCA runtime MUST treat an @org.oasisopen.sca.annotation.OneWay annotation as a synonym for the @javax.jws.OneWay annotation.

Deleted: For the WSDL-to-Java mapping, the SCA runtime MUST take the generated @WebService annotation to imply that the Java interface is @Remotable.

Deleted: For the WSDL-to-Java mapping, the SCA runtime MUST take the generated @WebService annotation to imply that the Java interface is @Remotable.

Deleted: March

2315 c. a return type of Future<?>
2316 then C is a JAX-WS callback method that isn't part of the SCA interface contract.
2317 As an example, an interface can be defined in WSDL as follows:

```
2318 <!-- WSDL extract -->  
2319 <message name="getPrice">  
2320 <part name="ticker" type="xsd:string"/>  
2321 </message>  
2322  
2323 <message name="getPriceResponse">  
2324 <part name="price" type="xsd:float"/>  
2325 </message>  
2326  
2327 <portType name="StockQuote">  
2328 <operation name="getPrice">  
2329 <input message="tns:getPrice"/>  
2330 <output message="tns:getPriceResponse"/>  
2331 </operation>  
2332 </portType>
```

2333
2334 The JAX-WS asynchronous mapping will produce the following Java interface:

```
2335 // asynchronous mapping  
2336 @WebService  
2337 public interface StockQuote {  
2338     float getPrice(String ticker);  
2339     Response<Float> getPriceAsync(String ticker);  
2340     Future<?> getPriceAsync(String ticker, AsyncHandler<Float>);  
2341 }
```

2342
2343 For SCA interface definition purposes, this is treated as equivalent to the following:

```
2344 // synchronous mapping  
2345 @WebService  
2346 public interface StockQuote {  
2347     float getPrice(String ticker);  
2348 }
```

2349
2350 **SCA runtimes MUST support the use of the JAX-WS client asynchronous model. [JCA100009]** In
2351 the above example, if the client implementation uses the asynchronous form of the interface, the
2352 two additional getPriceAsync() methods can be used for polling and callbacks as defined by the
2353 JAX-WS specification.

2354

A. XML Schema: sca-interface-java.xsd

```
2355 <?xml version="1.0" encoding="UTF-8"?>
2356 <!-- (c) Copyright SCA Collaboration 2006 -->
2357 <schema xmlns="http://www.w3.org/2001/XMLSchema"
2358         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2359         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2360         elementFormDefault="qualified">
2361
2362     <include schemaLocation="sca-core.xsd"/>
2363
2364     <element name="interface.java" type="sca:JavaInterface"
2365             substitutionGroup="sca:interface"/>
2366     <complexType name="JavaInterface">
2367         <complexContent>
2368             <extension base="sca:Interface">
2369                 <sequence>
2370                     <any namespace="##other" processContents="lax"
2371                         minOccurs="0" maxOccurs="unbounded"/>
2372                 </sequence>
2373                 <attribute name="interface" type="NCName" use="required"/>
2374                 <attribute name="callbackInterface" type="NCName"
2375                         use="optional"/>
2376                 <anyAttribute namespace="##any" processContents="lax"/>
2377             </extension>
2378         </complexContent>
2379     </complexType>
2380 </schema>
2381
```

2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430

B. Java Classes and Interfaces

B.1 SCAClient Classes and Interfaces

B.1.1 SCAClient Interface

```
/*  
 * Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.  
 * OASIS trademark, IPR and other policies apply.  
 */  
package org.oasisopen.sca.client;  
  
import org.oasisopen.sca.NoSuchDomainException;  
import org.oasisopen.sca.NoSuchServiceException;  
  
/**  
 * Client side interface that can be used to lookup SCA Services within  
 * a SCA domain.  
 * <p>  
 * The SCAClientFactory is used to obtain an implementation instance of  
 * the SCAClient.  
 *  
 * @see SCAClientFactory  
 * @author OASIS Open  
 */  
public interface SCAClient {  
  
    /**  
     * Returns a reference proxy that implements the business interface <T>  
     * of a service in a domain  
     *  
     * @param serviceURI the relative URI of the target service. Takes the  
     * form componentName/serviceName.  
     * Can also take the extended form componentName/serviceName/bindingName  
     * to use a specific binding of the target service  
     *  
     * @param interfaze The business interface class of the service in the  
     * domain  
     * @param <T> The business interface class of the service in the domain  
     *  
     * @return a proxy to the target service, in the specified SCA Domain  
     * that implements the business interface <B>.  
     * @throws NoSuchServiceException Service requested was not found  
     * @throws NoSuchDomainException Domain requested was not found  
     */  
    <T> T getService(Class<T> interfaze, String serviceURI)  
        throws NoSuchServiceException, NoSuchDomainException;  
}
```

Deleted: March

2431
2432

2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485

Deleted: D

B.1.2 SCAClientFactory Class

SCA provides an abstract base class SCAClientFactory. Vendors can provide subclasses of this class which create objects that implement the SCAClient interface suitable for linking to services in their SCA runtime.

```
/*
 * Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
 * OASIS trademark, IPR and other policies apply.
 */
package org.oasisopen.sca.client;

import java.net.URI;
import java.util.Properties;

import org.oasisopen.sca.client.SCAClientFactoryFinder;
import org.oasisopen.sca.client.impl.SCAClientFactoryFinderImpl;

/**
 * The SCAClientFactory can be used by non-SCA managed code to
 * lookup services that exist in a SCADomain.
 *
 * @see SCAClientFactoryFinderImpl
 * @see SCAClient
 *
 * @author OASIS Open
 */
public abstract class SCAClientFactory {

    /**
     * The SCAClientFactoryFinder.
     * Provides a means by which a provider of an SCAClientFactory
     * implementation can inject a factory finder implementation into
     * the abstract SCAClientFactory class - once this is done, future
     * invocations of the SCAClientFactory use the injected factory
     * finder to locate and return an instance of a subclass of
     * SCAClientFactory.
     */
    protected static SCAClientFactoryFinder factoryFinder;

    /**
     * The Domain URI of the SCA Domain which is accessed by this
     * SCAClientFactory
     */
    private URI domainURI;

    /**
     * Prevent concrete subclasses from using the no-arg constructor
     */
    private SCAClientFactory() {
    }

    /**
```

Formatted: French France

Formatted: Font color: Black

Formatted: Indent: First line: 0.5"

Deleted: March

```

2486     * Constructor used by concrete subclasses
2487     * @param domainURI - The Domain URI of the Domain accessed via this
2488     * SCAClientFactory
2489     */
2490     protected SCAClientFactory(URI domainURI) {
2491         this.domainURI = domainURI;
2492     }
2493
2494     /**
2495     * Gets the Domain URI of the Domain accessed via this SCAClientFactory
2496     * @return - the URI for the Domain
2497     */
2498     protected URI getDomainURI() {
2499         return domainURI;
2500     }
2501
2502
2503     /**
2504     * Creates a new instance of the SCAClient that can be
2505     * used to lookup SCA Services.
2506     *
2507     * @param domainURI      URI of the target domain for the SCAClient
2508     * @return A new SCAClient
2509     */
2510     public static SCAClient newInstance( URI domainURI ) {
2511         return newInstance(null, null, domainURI);
2512     }
2513
2514     /**
2515     * Creates a new instance of the SCAClient that can be
2516     * used to lookup SCA Services.
2517     *
2518     * @param properties     Properties that may be used when
2519     * creating a new instance of the SCAClient
2520     * @param domainURI      URI of the target domain for the SCAClient
2521     * @return A new SCAClient instance
2522     */
2523     public static SCAClient newInstance(Properties properties,
2524                                         URI domainURI) {
2525         return newInstance(properties, null, domainURI);
2526     }
2527
2528     /**
2529     * Creates a new instance of the SCAClient that can be
2530     * used to lookup SCA Services.
2531     *
2532     * @param classLoader    ClassLoader that may be used when
2533     * creating a new instance of the SCAClient
2534     * @param domainURI      URI of the target domain for the SCAClient
2535     * @return A new SCAClient instance
2536     */
2537     public static SCAClient newInstance(ClassLoader classLoader,
2538                                         URI domainURI) {
2539         return newInstance(null, classLoader, domainURI);
2540     }
2541
2542     /**
2543     * Creates a new instance of the SCAClient that can be

```

Deleted: March


```

2544     * used to lookup SCA Services.
2545     *
2546     * @param properties    Properties that may be used when
2547     * creating a new instance of the SCAClient
2548     * @param classLoader  ClassLoader that may be used when
2549     * creating a new instance of the SCAClient
2550     * @param domainURI    URI of the target domain for the SCAClient
2551     * @return A new SCAClient instance
2552     */
2553     public static SCAClient newInstance(Properties properties,
2554                                       ClassLoader classLoader,
2555                                       URI domainURI) {
2556         final SCAClientFactoryFinder finder =
2557             factoryFinder != null ? factoryFinder :
2558             new SCAClientFactoryFinderImpl();
2559         final SCAClientFactory factory
2560             = finder.find(properties, classLoader, domainURI);
2561         return factory.createSCAClient();
2562     }
2563
2564     /**
2565     * This method is invoked to create a new SCAClient instance.
2566     *
2567     * @return A new SCAClient instance
2568     */
2569     protected abstract SCAClient createSCAClient();
2570 }
2571
2572

```

2573 B.1.3 SCAClientFactoryFinder interface

2574 The SCAClientFactoryFinder interface is a Service Provider Interface representing a SCAClientFactory
2575 finder. SCA provides a default reference implementation of this interface. SCA runtime vendors can
2576 create alternative implementations of this interface that use different class loading or lookup mechanisms.

```

2577     /*
2578     * Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
2579     * OASIS trademark, IPR and other policies apply.
2580     */
2581
2582     package org.oasisopen.sca.client;
2583
2584     import java.net.URI;
2585     import java.util.Properties;
2586
2587     /* A Service Provider Interface representing a SCAClientFactory finder.
2588     * SCA provides a default reference implementation of this interface.
2589     * SCA runtime vendors can create alternative implementations of this
2590     * interface that use different class loading or lookup mechanisms.
2591     */
2592     public interface SCAClientFactoryFinder {
2593
2594         /**
2595         * Method for finding the SCAClientFactory for a given Domain URI using
2596         * a specified set of properties and a a specified ClassLoader
2597         * @param properties - properties to use - may be null
2598         * @param classLoader - ClassLoader to use - may be null
2599

```

Deleted: protected

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

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

Formatted: Bullets and
Numbering

Formatted: Normal

Formatted: French France

Deleted: March

```

2600     * @param domainURI - the Domain URI - must be a valid SCA Domain URI
2601     * @return - the SCAClientFactory or null if the factory could not be
2602     * found
2603     */
2604     SCAClientFactory find(Properties properties,
2605                           ClassLoader classLoader,
2606                           URI domainURI );
2607 }

```

Formatted: Normal

Formatted: Bullets and Numbering

2608 **B.1.4 SCAClientFactoryFinderImpl class**

2609 This class provides a default implementation for finding a provider's SCAClientFactory implementation
2610 class. It is used if the provider does not inject its SCAClientFactoryFinder implementation class into the
2611 base SCAClientFactory class.

2612 It discovers a provider's SCAClientFactory implementation by referring to the following information in this
2613 order:

- 2614 1. [The org.oasisopen.sca.client.SCAClientFactory property from the Properties specified on the](#)
2615 [newInstance\(\) method call if specified](#)
- 2616 2. [The org.oasisopen.sca.client.SCAClientFactory property from the System Properties](#)
- 2617 3. [The META-INF/services/org.oasisopen.sca.client.SCAClientFactory file](#)

```

2618 /*
2619  * Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
2620  * OASIS trademark, IPR and other policies apply.
2621  */

```

Deleted: Since this is a reference implementation, vendors are free to replace the SCAClientFactoryFinder class with an alternative implementation that provides the lookup mechanisms required for their SCA Runtime.

```

2622 package org.oasisopen.sca.client.impl;
2623
2624 import org.oasisopen.sca.client.SCAClientFactoryFinder;
2625
2626 import java.io.BufferedReader;
2627 import java.io.Closeable;
2628 import java.io.IOException;
2629 import java.io.InputStream;
2630 import java.io.InputStreamReader;
2631 import java.lang.reflect.Constructor;
2632 import java.net.URI;
2633 import java.net.URL;
2634 import java.util.Properties;
2635
2636 import org.oasisopen.sca.ServiceRuntimeException;
2637 import org.oasisopen.sca.client.SCAClientFactory;

```

Formatted: French France

```

2638 /**
2639  * This is a default implementation of an SCAClientFactoryFinder which is
2640  * used to find an implementation of the SCAClientFactory interface.
2641  *
2642  * @see SCAClientFactoryFinder
2643  * @see SCAClientFactory
2644  *
2645  * @author OASIS Open
2646  */
2647
2648 public class SCAClientFactoryFinderImpl implements SCAClientFactoryFinder {
2649
2650     /**
2651     * The name of the System Property used to determine the SPI
2652     * implementation to use for the SCAClientFactory.
2653     */

```

Deleted: March

```

2654 private static final String SCA_CLIENT_FACTORY_PROVIDER_KEY =
2655 SCAClientFactory.class.getName();
2656
2657 /**
2658 * The name of the file loaded from the ClassPath to determine
2659 * the SPI implementation to use for the SCAClientFactory.
2660 */
2661 private static final String SCA_CLIENT_FACTORY_PROVIDER_META_INF_SERVICE
2662 = "META-INF/services/" + SCA_CLIENT_FACTORY_PROVIDER_KEY;
2663
2664 /**
2665 * Public Constructor
2666 */
2667 public SCAClientFactoryFinderImpl() {
2668 }
2669
2670 /**
2671 * Creates an instance of the SCAClientFactorySPI implementation.
2672 * This discovers the SCAClientFactorySPI Implementation and instantiates
2673 * the provider's implementation.
2674 *
2675 * @param properties Properties that may be used when creating a new
2676 * instance of the SCAClient
2677 * @param classLoader ClassLoader that may be used when creating a new
2678 * instance of the SCAClient
2679 * @return new instance of the SCAClientFactory
2680 * @throws ServiceRuntimeException Failed to create SCAClientFactory
2681 * Implementation.
2682 */
2683 public SCAClientFactory find(Properties properties,
2684 ClassLoader classLoader,
2685 URI domainURI )
2686 throws ServiceRuntimeException {
2687 if (classLoader == null) {
2688 classLoader = getThreadContextClassLoader();
2689 }
2690 final String factoryImplClassName =
2691 discoverProviderFactoryImplClass(properties, classLoader);
2692 final Class<? extends SCAClientFactory> factoryImplClass
2693 = loadProviderFactoryClass(factoryImplClassName,
2694 classLoader);
2695 final SCAClientFactory factory =
2696 instantiatesCAClientFactoryClass(factoryImplClass,
2697 domainURI
2698 );
2699 return factory;
2700 }
2701
2702 /**
2703 * Gets the Context ClassLoader for the current Thread.
2704 *
2705 * @return The Context ClassLoader for the current Thread.
2706 */
2707 private static ClassLoader getThreadContextClassLoader() {
2708 final ClassLoader threadClassLoader =
2709 Thread.currentThread().getContextClassLoader();
2710 return threadClassLoader;
2711 }

```

Deleted: March

```

2712
2713 /**
2714 * Attempts to discover the class name for the SCAClientFactorySPI
2715 * implementation from the specified Properties, the System Properties
2716 * or the specified ClassLoader.
2717 *
2718 * @return The class name of the SCAClientFactorySPI implementation
2719 * @throw ServiceRuntimeException Failed to find implementation for
2720 * SCAClientFactorySPI.
2721 */
2722 private static String
2723 discoverProviderFactoryImplClass(Properties properties,
2724 ClassLoader classLoader)
2725 throws ServiceRuntimeException {
2726 String providerClassName =
2727 checkPropertiesForSPIClassName(properties);
2728 if (providerClassName != null) {
2729 return providerClassName;
2730 }
2731
2732 providerClassName =
2733 checkPropertiesForSPIClassName(System.getProperties());
2734 if (providerClassName != null) {
2735 return providerClassName;
2736 }
2737
2738 providerClassName = checkMETAINFOServicesForSIPClassName(classLoader);
2739 if (providerClassName == null) {
2740 throw new ServiceRuntimeException(
2741 "Failed to find implementation for SCAClientFactory");
2742 }
2743
2744 return providerClassName;
2745 }
2746
2747 /**
2748 * Attempts to find the class name for the SCAClientFactorySPI
2749 * implementation from the specified Properties.
2750 *
2751 * @return The class name for the SCAClientFactorySPI implementation
2752 * or <code>null</code> if not found.
2753 */
2754 private static String
2755 checkPropertiesForSPIClassName(Properties properties) {
2756 if (properties == null) {
2757 return null;
2758 }
2759
2760 final String providerClassName =
2761 properties.getProperty(SCA_CLIENT_FACTORY_PROVIDER_KEY);
2762 if (providerClassName != null && providerClassName.length() > 0) {
2763 return providerClassName;
2764 }
2765
2766 return null;
2767 }
2768
2769 /**

```

```

2770     * Attempts to find the class name for the SCAClientFactorySPI
2771     * implementation from the META-INF/services directory
2772     *
2773     * @return The class name for the SCAClientFactorySPI implementation or
2774     * <code>null</code> if not found.
2775     */
2776     private static String checkMETAINFServicesForSIPClassName(ClassLoader cl)
2777     {
2778         final URL url =
2779             cl.getResource(SCA_CLIENT_FACTORY_PROVIDER_META_INF_SERVICE);
2780         if (url == null) {
2781             return null;
2782         }
2783
2784         InputStream in = null;
2785         try {
2786             in = url.openStream();
2787             BufferedReader reader = null;
2788             try {
2789                 reader =
2790                     new BufferedReader(new InputStreamReader(in, "UTF-8"));
2791
2792                 String line;
2793                 while ((line = readNextLine(reader)) != null) {
2794                     if (!line.startsWith("#") && line.length() > 0) {
2795                         return line;
2796                     }
2797                 }
2798
2799                 return null;
2800             } finally {
2801                 closeStream(reader);
2802             }
2803         } catch (IOException ex) {
2804             throw new ServiceRuntimeException(
2805                 "Failed to discover SCAClientFactory provider", ex);
2806         } finally {
2807             closeStream(in);
2808         }
2809     }
2810
2811     /**
2812     * Reads the next line from the reader and returns the trimmed version
2813     * of that line
2814     *
2815     * @param reader The reader from which to read the next line
2816     * @return The trimmed next line or <code>null</code> if the end of the
2817     * stream has been reached
2818     * @throws IOException I/O error occurred while reading from Reader
2819     */
2820     private static String readNextLine(BufferedReader reader)
2821     throws IOException {
2822
2823         String line = reader.readLine();
2824         if (line != null) {
2825             line = line.trim();
2826         }
2827         return line;

```

```

2828     }
2829
2830     /**
2831     * Loads the specified SCAClientFactory Implementation class.
2832     *
2833     * @param factoryImplClassName The name of the SCAClientFactory
2834     * Implementation class to load
2835     * @return The specified SCAClientFactory Implementation class
2836     * @throws ServiceRuntimeException Failed to load the SCAClientFactory
2837     * Implementation class
2838     */
2839     private static Class<? extends SCAClientFactory>
2840     loadProviderFactoryClass(String factoryImplClassName,
2841                             ClassLoader classLoader)
2842     throws ServiceRuntimeException {
2843
2844         try {
2845             final Class<?> providerClass =
2846                 classLoader.loadClass(factoryImplClassName);
2847             final Class<? extends SCAClientFactory> providerFactoryClass =
2848                 providerClass.asSubclass(SCAClientFactory.class);
2849             return providerFactoryClass;
2850         } catch (ClassNotFoundException ex) {
2851             throw new ServiceRuntimeException(
2852                 "Failed to load SCAClientFactory implementation class "
2853                 + factoryImplClassName, ex);
2854         } catch (ClassCastException ex) {
2855             throw new ServiceRuntimeException(
2856                 "Loaded SCAClientFactory implementation class "
2857                 + factoryImplClassName
2858                 + " is not a subclass of "
2859                 + SCAClientFactory.class.getName() , ex);
2860         }
2861     }
2862
2863     /**
2864     * Instantiate an instance of the specified SCAClientFactorySPI
2865     * Implementation class.
2866     *
2867     * @param factoryImplClass The SCAClientFactorySPI Implementation
2868     * class to instantiate.
2869     * @return An instance of the SCAClientFactorySPI Implementation class
2870     * @throws ServiceRuntimeException Failed to instantiate the specified
2871     * specified SCAClientFactorySPI Implementation class
2872     */
2873     private static SCAClientFactory instantiateSCAClientFactoryClass(
2874         Class<? extends SCAClientFactory> factoryImplClass,
2875         URI domainURI)
2876     throws ServiceRuntimeException {
2877
2878         try {
2879             Constructor<? extends SCAClientFactory> URIConstructor =
2880                 factoryImplClass.getConstructor(domainURI.getClass());
2881             SCAClientFactory provider = URIConstructor.newInstance( domainURI
2882 );
2883             return provider;
2884         } catch (Throwable ex) {
2885             throw new ServiceRuntimeException(

```

Deleted: March

```

2886     "Failed to instantiate SCAClientFactory implementation class "
2887     + factoryImplClass, ex);
2888 }
2889 }
2890
2891 /**
2892  * Utility method for closing Closeable Object.
2893  *
2894  * @param closeable The Object to close.
2895  */
2896 private static void closeStream(Closeable closeable) {
2897     if (closeable != null) {
2898         try{
2899             closeable.close();
2900         } catch (IOException ex) {
2901             throw new ServiceRuntimeException("Failed to close stream",
2902                                             ex);
2903         }
2904     }
2905 }
2906 }
2907

```

Formatted: Bullets and Numbering

2908 **B.1.5 SCAClient Classes and Interfaces - what does a vendor need to do?**

2909 The SCAClient classes and interfaces are designed so that vendors can provide their own
2910 implementation suited to the needs of their SCA runtime. This section describes the tasks that a vendor
2911 needs to consider in relation to the SCAClient classes and interfaces.

- 2912 • Implement their SCAClientFactory and SCAClient implementation classes

2914 Vendors need to provide an implementation of SCAClient that is capable of looking up Services in
2915 their SCA Runtime.

2917 Vendors need to subclass SCAClientFactory and implement the createSCAClient() method so
2918 that it creates an instance of their SCAClient implementation.

- 2921 • Configure the Vendor Implementation classes so they are used
2922 Vendors have several options:

2924 Option 1: Set System Property to point to the Vendor's implementation

2926 Vendors set the org.oasisopen.sca.client.SCAClientFactory System Property to point to their
2927 implementation class and use the reference implementation of SCAClientFactoryFinder

2929 Option 2: Provide a META-INF/services file

2931 Vendors provide a META-INF/services/org.oasisopen.sca.client.SCAClientFactory file that points
2932 to their implementation class and use the reference implementation of SCAClientFactoryFinder

2934 Option 3: Inject a vendor implementation of the SCAClientFactoryFinder interface into
2935 SCAClientFactory

2937 Vendors inject an instance into the factoryFinder field of SCAClientFactory. The reference
2938 implementation of SCAClientFactoryFinder is not used in this scenario.

Deleted: -
Option 4: Provide a Vendor
specific implementation of
SCAClientFactoryFinder -
-
Vendors write a new
implementation of
SCAClientFactoryFinder and
replace the reference
implementation that is provided
by SCA.

Deleted: March

2940

C. Conformance Items

2941 This section contains a list of conformance items for the SCA Java Common Annotations and APIs
2942 specification.

2943

Conformance ID	Description
[JCA20001]	Remotable Services MUST NOT make use of method overloading .
[JCA20002]	the SCA runtime MUST ensure that a stateless scoped implementation instance object is only ever dispatched on one thread at any one time.
[JCA20003]	within the SCA lifecycle of a stateless scoped implementation instance, the SCA runtime MUST only make a single invocation of one business method.
[JCA20004]	For a composite scope implementation instance, the SCA runtime MUST ensure that all service requests are dispatched to the same implementation instance for the lifetime of the containing composite.
[JCA20005]	When the implementation class is marked for eager initialization, the SCA runtime MUST create a composite scoped instance when its containing component is started.
[JCA20006]	If a method of an implementation class is marked with the @Init annotation, the SCA runtime MUST call that method when the implementation instance is created.
[JCA20007]	the SCA runtime MAY run multiple threads in a single composite scoped implementation instance object and the SCA runtime MUST NOT perform any synchronization.
[JCA30001]	The value of the @interface attribute MUST be the fully qualified name of the Java interface class
[JCA30002]	The value of the @callbackInterface attribute MUST be the fully qualified name of a Java interface used for callbacks
[JCA30003]	if the Java interface class identified by the @interface attribute does contain a Java @Callback annotation, then the Java interface class identified by the @callbackInterface attribute MUST be the same interface class.
[JCA30004]	The interface.java element MUST conform to the schema defined in the sca-interface-java.xsd schema.
[JCA70001]	SCA identifies annotations that correspond to intents by providing an @Intent annotation which MUST be used in the definition of a specific intent annotation.
[JCA80001]	ComponentContext.getService method MUST throw an IllegalArgumentException if the reference identified by the referenceName parameter has multiplicity of 0..n or 1..n.

- [JCA80002] The ComponentContext.getRequestContext method MUST return non-null when invoked during the execution of a Java business method for a service operation or a callback operation, on the same thread that the SCA runtime provided, and MUST return null in all other cases.
- [JCA80003] When invoked during the execution of a service operation, the getServiceReference method MUST return a ServiceReference that represents the service that was invoked. When invoked during the execution of a callback operation, the getServiceReference method MUST return a ServiceReference that represents the callback that was invoked.
- [JCA90001] An SCA runtime MUST verify the proper use of all SCA annotations and if an annotation is improperly used, the SCA runtime MUST NOT run the component which uses the invalid implementation code.
- [JCA90002] SCA annotations MUST NOT be used on static methods or on static fields. It is an error to use an SCA annotation on a static method or a static field of an implementation class and the SCA runtime MUST NOT instantiate such an implementation class.
- [JCA90003] If a constructor of an implementation class is annotated with @Constructor and the constructor has parameters, each of these parameters MUST have either a @Property annotation or a @Reference annotation.
- [JCA90004] A method annotated with @Destroy MAY have any access modifier and MUST have a void return type and no arguments.
- [JCA90005] If there is a method annotated with @Destroy that matches the criteria for the annotation, the SCA runtime MUST call the annotated method when the scope defined for the implementation class ends.
- [JCA90007] When marked for eager initialization with an @EagerInit annotation, the composite scoped instance MUST be created when its containing component is started.
- [JCA90008] A method marked with the @Init annotation MAY have any access modifier and MUST have a void return type and no arguments.
- [JCA90009] If there is a method annotated with @Init that matches the criteria for the annotation, the SCA runtime MUST call the annotated method after all property and reference injection is complete.
- [JCA90011] the @Property annotation MUST NOT be used on a class field that is declared as final.
- [JCA90012] the @Property annotation MUST be used in order to inject a property onto a non-public field.
- [JCA90013] For a @Property annotation applied to a constructor parameter, there is no default value for the name attribute and the name attribute MUST be present.

- [JCA90014] For a @Property annotation applied to a constructor parameter, the required attribute MUST have the value true.
- [JCA90015] The @Qualifier annotation MUST be used in a specific intent annotation definition where the intent has qualifiers.
- [JCA90016] The @Reference annotation MUST NOT be used on a class field that is declared as final.
- [JCA90017] the @Reference annotation MUST be used in order to inject a reference onto a non-public field.
- [JCA90018] For a @Reference annotation applied to a constructor parameter, there is no default for the name attribute and the name attribute MUST be present.
- [JCA90019] For a @Reference annotation applied to a constructor parameter, the required attribute MUST have the value true.
- [JCA90020] If the type of a reference is not an array or any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference/> element with @multiplicity= 0..1 if the @Reference annotation required attribute is false and with @multiplicity=1..1 if the @Reference annotation required attribute is true.
- [JCA90021] If the type of a reference is defined as an array or as any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference/> element with @multiplicity=0..n if the @Reference annotation required attribute is false and with @multiplicity=1..n if the @Reference annotation required attribute is true.
- [JCA90022] An unwired reference with a multiplicity of 0..1 MUST be presented to the implementation code by the SCA runtime as null (either via injection or via API call).
- [JCA90023] An unwired reference with a multiplicity of 0..n MUST be presented to the implementation code by the SCA runtime as an empty array or empty collection (either via injection or via API call).
- [JCA90024] References MAY be reinjected by an SCA runtime after the initial creation of a component if the reference target changes due to a change in wiring that has occurred since the component was initialized.
- [JCA90025] In order for reinjection to occur, the following MUST be true:
1. The component MUST NOT be STATELESS scoped.
 2. The reference MUST use either field-based injection or setter injection. References that are injected through constructor injection MUST NOT be changed.
- [JCA90026] If a reference target changes and the reference is not reinjected, the reference MUST continue to work as if the reference target was not changed.
- [JCA90027] If an operation is called on a reference where the target of that reference has been undeployed, the SCA runtime SHOULD throw an InvalidServiceException.

Deleted: March

[JCA90028]

If an operation is called on a reference where the target of the reference has become unavailable for some reason, the SCA runtime SHOULD throw a ServiceUnavailableException.

[JCA90029]

If the target service of the reference is changed, the reference MAY continue to work, depending on the runtime and the type of change that was made.

[JCA90030]

A ServiceReference that has been obtained from a reference by ComponentContext.cast() corresponds to the reference that is passed as a parameter to cast(). If the reference is subsequently reinjected, the ServiceReference obtained from the original reference MUST continue to work as if the reference target was not changed.

[JCA90031]

If the target of a ServiceReference has been undeployed, the SCA runtime SHOULD throw a InvalidServiceException when an operation is invoked on the ServiceReference.

Deleted: [JCA90031]

Deleted: [JCA90031]

[JCA90032]

If the target of a ServiceReference has become unavailable, the SCA runtime SHOULD throw a ServiceUnavailableException when an operation is invoked on the ServiceReference.

Deleted: [JCA90032]

Deleted: [JCA90032]

[JCA90033]

If the target service of a ServiceReference is changed, the reference MAY continue to work, depending on the runtime and the type of change that was made.

Deleted: [JCA90033]

Deleted: [JCA90033]

[JCA90034]

A reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() MUST correspond to the current configuration of the domain. This applies whether or not reinjection has taken place.

Deleted: [JCA90034]

Deleted: [JCA90034]

[JCA90035]

If the target of a reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() has been undeployed or has become unavailable, the result SHOULD be a reference to the undeployed or unavailable service, and attempts to call business methods SHOULD throw an InvalidServiceException or a ServiceUnavailableException.

Deleted: [JCA90035]

Deleted: [JCA90035]

[JCA90036]

If the target service of a reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() has changed, the returned value SHOULD be a reference to the changed service.

Deleted: [JCA90036]

Deleted: [JCA90036]

[JCA90037]

in the cases where reference reinjection is not allowed, the array or Collection for a reference of multiplicity 0..n or multiplicity 1..n MUST NOT change its contents when changes occur to the reference wiring or to the targets of the wiring.

Deleted: [JCA90037]

Deleted: [JCA90037]

[JCA90038]

In cases where the contents of a reference array or collection change when the wiring changes or the targets change, then for references that use setter injection, the setter method MUST be called by the SCA runtime for any change to the contents.

Deleted: [JCA90038]

Deleted: [JCA90038]

[JCA90039]

A reinjected array or Collection for a reference MUST NOT be the same array or Collection object previously injected to the component.

Deleted: [JCA90039]

Deleted: [JCA90039]

Deleted: March

[JCA90040]

The @Remotable annotation is used to specify a Java service interface as remotable. A remotable service can be published externally as a service and MUST be translatable into a WSDL portType.

[JCA90041]

The @Scope annotation MUST only be used on a service's implementation class. It is an error to use this annotation on an interface.

[JCA90042]

An implementation class need not be declared as implementing all of the interfaces implied by the services declared in its @Service annotation, but all methods of all the declared service interfaces MUST be present.

Formatted: Pattern: Clear (Light Yellow)

[JCA90043]

A @Service annotation MUST only have one of the interfaces attribute or value attribute specified.

[JCA90044]

A @Service annotation with no attributes MUST be ignored, it is the same as not having the annotation there at all.

[JCA90045]

A component implementation MUST NOT have two services with the same Java simple name.

Deleted: [JCA90045]

Deleted: [JCA90045]

[JCA90046]

When used to annotate a method or a field of an implementation class for injection of a callback object, the @Callback annotation MUST NOT specify any attributes.

[JCA90047]

For a @Property annotation, if the the type of the Java class field or the type of the input parameter of the setter method or constructor is defined as an array or as any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <property/> element with a @many attribute set to true, otherwise @many MUST be set to false.

[JCA100001]

For the purposes of the Java-to-WSDL mapping algorithm, the SCA runtime MUST treat a Java interface as if it had a @WebService annotation on the class, even if it doesn't.

[JCA100002]

The SCA runtime MUST treat an @org.oasisopen.sca.annotation.OneWay annotation as a synonym for the @javax.jws.OneWay annotation.

[JCA100003]

For the WSDL-to-Java mapping, the SCA runtime MUST take the generated @WebService annotation to imply that the Java interface is @Remotable.

[JCA100004]

SCA runtimes MUST support the JAXB 2.1 mapping from Java types to XML schema types.

[JCA100005]

SCA runtimes MAY support the SDO 2.1 mapping from Java types to XML schema types.

[JCA100006]

For SCA service interfaces defined using interface.java, the Java interface MUST NOT contain the additional client-side asynchronous polling and callback methods defined by JAX-WS.

[JCA100007]

For SCA reference interfaces defined using interface.java, the Java interface MAY contain the additional client-side asynchronous polling and callback methods defined by JAX-WS.

Deleted: March

[JCA100008]

If the additional client-side asynchronous polling and callback methods defined by JAX-WS are present in the interface which declares the type of a reference in the implementation, SCA Runtimes MUST NOT include these methods in the SCA reference interface in the component type of the implementation.

[JCA100009]

SCA runtimes MUST support the use of the JAX-WS client asynchronous model.

2944

2945 **D. Acknowledgements**

2946 The following individuals have participated in the creation of this specification and are gratefully
2947 acknowledged:

2948 **Participants:**

2949 [Participant Name, Affiliation | Individual Member]

2950 [Participant Name, Affiliation | Individual Member]

2951

Deleted: March

E. Non-Normative Text

Deleted: March

2953

F. Revision History

2954 [optional; should not be included in OASIS Standards]

2955

Revision	Date	Editor	Changes Made
1	2007-09-26	Anish Karmarkar	Applied the OASIS template + related changes to the Submission
2	2008-02-28	Anish Karmarkar	Applied resolution of issues: 4, 11, and 26
3	2008-04-17	Mike Edwards	Ed changes
4	2008-05-27	Anish Karmarkar David Booz Mark Combella	Added InvalidServiceException in Section 7 Various editorial updates
WD04	2008-08-15	Anish Karmarkar	* Applied resolution of issue 9 (it was applied before, not sure by whom, but it was applied incorrectly) * Applied resolution of issue 12, 22, 23, 29, 31, 35, 36, 37, 44, 45 * Note that issue 33 was applied, but not noted, in a previous version * Replaced the osoa.org NS with the oasis-open.org NS
WD05	2008-10-03	Anish Karmarkar	* Fixed the resolution of issue 37 but re-adding the sentence: "However, the @... annotation must be used in order to inject a property onto a non-public field. -- in the @Property and @Reference section * resolution of issue 9 was applied incorrectly. Fixed that -- removed the requirement for throwing an exception on ComponentContext.getServiceReferences() when multiplicity of references > 1 * minor ed changes
cd01-rev1	2008-12-11	Anish Karmarkar	* Fixed reference style to [RFC2119] instead of [1]. * Applied resolutions of issues 20, 21, 41, 42, 43, 47, 48, 49.
cd01-rev2	2008-12-12	Anish Karmarkar	* Applied resolutions of issues 61, 71, 72, 73, 79, 81, 82, 84, 112
cd01-rev3	2008-12-16	David Booz	* Applied resolution of issues 56, 75, 111
cd01-rev4	2009-01-18	Anish Karmarkar	* Applied resolutions of issues 28, 52, 94, 96, 99, 101
cd02	2009-01-26	Mike Edwards	Minor editorial cleanup. All changes accepted.

Deleted: March

			All comments removed.
cd02-rev1	2009-02-03	Mike Edwards	Issues 25+95 Issue 120
cd02-rev2	2009-02-08	Mike Edwards	Merge annotation definitions contained in section 10 into section 8 Move remaining parts of section 10 to section 7. Accept all changes.
cd02-rev3	2009-03-16	Mike Edwards	RFC2119 work and formal marking of all normative statements - all sections. Completion of Appendix B (list of all normative statements) Accept all changes

2956

Page 52: [1] Deleted	Mike Edwards	6/23/2009 9:55:00 AM
<p>If the type of a reference is not an array or any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference/> element with @multiplicity= 0..1 if the @Reference annotation required attribute is false and with @multiplicity=1..1 if the @Reference annotation required attribute is true.</p>		
Page 52: [2] Deleted	Mike Edwards	6/23/2009 9:55:00 AM
<p>If the type of a reference is defined as an array or as any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference/> element with @multiplicity=0..n if the @Reference annotation required attribute is false and with @multiplicity=1..n if the @Reference annotation required attribute is true.</p>		
Page 52: [3] Deleted	Mike Edwards	6/23/2009 9:55:00 AM
<p>If the type of a reference is defined as an array or as any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference/> element with @multiplicity=0..n if the @Reference annotation required attribute is false and with @multiplicity=1..n if the @Reference annotation required attribute is true.</p>		
Page 52: [4] Deleted	Mike Edwards	6/23/2009 9:55:00 AM
<p>An unwired reference with a multiplicity of 0..n MUST be presented to the implementation code by the SCA runtime as an empty array or empty collection</p>		
Page 52: [5] Deleted	Mike Edwards	6/23/2009 9:55:00 AM
<p>An unwired reference with a multiplicity of 0..n MUST be presented to the implementation code by the SCA runtime as an empty array or empty collection</p>		
Page 53: [6] Deleted	Mike Edwards	6/23/2009 9:55:00 AM
<p>If the target service of the reference is changed, the reference MAY continue to work, depending on the runtime and the type of change that was made.</p>		
Page 53: [7] Deleted	Mike Edwards	6/23/2009 9:55:00 AM
<p>If the target service of the reference is changed, the reference MAY continue to work, depending on the runtime and the type of change that was made.</p>		
Page 53: [8] Deleted	Mike Edwards	6/23/2009 9:55:00 AM
<p>If the target service of a ServiceReference is changed, the reference MAY continue to work, depending on the runtime and the type of change that was made.</p>		
Page 53: [9] Deleted	Mike Edwards	6/23/2009 9:55:00 AM
<p>If the target service of a ServiceReference is changed, the reference MAY continue to work, depending on the runtime and the type of change that was made.</p>		
Page 53: [10] Deleted	Mike Edwards	6/23/2009 9:55:00 AM
<p>A reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() MUST correspond to the current configuration of the domain. This applies whether or not reinjection has taken place.</p>		
Page 53: [11] Deleted	Mike Edwards	6/23/2009 9:55:00 AM

A reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() MUST correspond to the current configuration of the domain. This applies whether or not reinjection has taken place.

Page 53: [12] Deleted **Mike Edwards** **6/23/2009 9:55:00 AM**

If the target of a reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() has been undeployed or has become unavailable, the result SHOULD be a reference to the undeployed or unavailable service, and attempts to call business methods SHOULD throw an InvalidServiceException or a ServiceUnavailableException.

Page 53: [13] Deleted **Mike Edwards** **6/23/2009 9:55:00 AM**

If the target of a reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() has been undeployed or has become unavailable, the result SHOULD be a reference to the undeployed or unavailable service, and attempts to call business methods SHOULD throw an InvalidServiceException or a ServiceUnavailableException.

Page 53: [14] Deleted **Mike Edwards** **6/23/2009 9:55:00 AM**

If the target service of a reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() has changed, the returned value SHOULD be a reference to the changed service.

Page 53: [15] Deleted **Mike Edwards** **6/23/2009 9:55:00 AM**

If the target service of a reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() has changed, the returned value SHOULD be a reference to the changed service.

Page 53: [16] Deleted **Mike Edwards** **6/23/2009 9:55:00 AM**

In cases where the contents of a reference array or collection change when the wiring changes or the targets change, then for references that use setter injection, the setter method MUST be called by the SCA runtime for any change to the contents.

Page 53: [17] Deleted **Mike Edwards** **6/23/2009 9:55:00 AM**

In cases where the contents of a reference array or collection change when the wiring changes or the targets change, then for references that use setter injection, the setter method MUST be called by the SCA runtime for any change to the contents.