



Service Component Architecture Java Common Annotations and APIs Specification Version 1.1

Committee Draft 02, Revision 03+Issue1 rev 9

30 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	16
6	Asynchronous Programming	17
6.1	@OneWay	17
6.2	Callbacks	17
6.2.1	Using Callbacks	17
6.2.2	Callback Instance Management	19
6.2.3	Implementing Multiple Bidirectional Interfaces	19
6.2.4	Accessing Callbacks	20
7	Policy Annotations for Java	21
7.1	General Intent Annotations	21
7.2	Specific Intent Annotations	23
7.2.1	How to Create Specific Intent Annotations	23
7.3	Application of Intent Annotations	24
7.3.1	Inheritance And Annotation	24
7.4	Relationship of Declarative And Annotated Intents	26
7.5	Policy Set Annotations	26
7.6	Security Policy Annotations	27
7.6.1	Security Interaction Policy	27
7.6.2	Security Implementation Policy	28
8	Java API	31

8.1 Component Context.....	31	
8.2 Request Context.....	32	
8.3 ServiceReference.....	33	
8.4 ServiceRuntimeException.....	33	
8.5 ServiceUnavailableException.....	34	
8.6 InvalidServiceException.....	34	
8.7 Constants.....	34	
8.8 SCAClient Interface.....	Error! Bookmark not defined.	
8.9 SCAClientFactory Class.....	34	Deleted: 36
8.10 NoSuchDomainException.....	38	Deleted: 37
8.11 NoSuchServiceException.....	39	Deleted: 37
9 Java Annotations.....	40	Deleted: 37
9.1 @AllowsPassByReference.....	40	Deleted: 37
9.2 @Authentication.....	40	Deleted: 38
9.3 @Callback.....	41	Deleted: 39
9.4 @ComponentName.....	42	Deleted: 40
9.5 @Confidentiality.....	43	Deleted: 40
9.6 @Constructor.....	44	Deleted: 41
9.7 @Context.....	44	Deleted: 41
9.8 @Destroy.....	45	Deleted: 42
9.9 @EagerInit.....	45	Deleted: 42
9.10 @Init.....	46	Deleted: 43
9.11 @Integrity.....	46	Deleted: 43
9.12 @Intent.....	47	Deleted: 44
9.13 @OneWay.....	48	Deleted: 45
9.14 @PolicySets.....	48	Deleted: 45
9.15 @Property.....	49	Deleted: 46
9.16 @Qualifier.....	50	Deleted: 47
9.17 @Reference.....	51	Deleted: 48
9.17.1 Reinjection.....	53	Deleted: 50
9.18 @Remotable.....	55	Deleted: 52
9.19 @Requires.....	56	Deleted: 53
9.20 @Scope.....	57	Deleted: 54
9.21 @Service.....	58	Deleted: 55
10 WSDL to Java and Java to WSDL.....	60	Deleted: 57
10.1 JAX-WS Client Asynchronous API for a Synchronous Service.....	60	Deleted: 57
A. XML Schema: sca-interface-java.xsd.....	62	Deleted: 59
B. Java Classes and Interfaces.....	63	Deleted: 60
B.1 SCAClient Classes and Interfaces.....	63	Deleted: 60
B.1.1 SCAClient Interface.....	Error! Bookmark not defined.	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.....	73	Deleted: 69
D. Acknowledgements.....	79	Deleted: 75
		Deleted: March

E. Non-Normative Text 80
F. Revision History..... 81

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>

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.

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

```
232 package org.oasisopen.sca.client.example;
```

```
233 import java.net.URI;
```

```
234 import org.oasisopen.sca.client.SCAClientFactory;
```

```
235 import org.oasisopen.sca.client.example.HelloService;
```

```
236 /**
```

```
237 * Example of use of Client API for a client application to obtain
```

```
238 * an SCA reference proxy for a service in an SCA Domain.
```

```
239 */
```

```
240 public class Client1 {
```

```
241     public void someMethod() {
```

```
242         try {
```

```
243             String serviceURI = "SomeHelloServiceURI";
```

```
244             URI domainURI = new URI("SomeDomainURI");
```

```
245             SCAClientFactory scaClient =
```

```
246                 SCAClientFactory.newInstance( domainURI );
```

```
247             HelloService helloService =
```

```
248                 scaClient.getService(HelloService.class,
```

```
249                 scaClient.getService(HelloService.class,
```

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

Formatted: Indent: Before: 0.25"

Formatted: Indent: Before: 1.25", First line: 0.25"

Formatted: Indent: Before: 0.25"

Deleted: March

```

257         serviceURI);
258     String reply = helloService.sayHello("Mark");
259
260     } catch (Exception e) {
261         System.out.println("Received exception");
262     }
263 }
264 }
265

```

266 [For details about the SCAClient interface and its related classes see the section "SCAClient](#)
267 [Interface" and the section "SCAClientFactory Class"](#).

268

Deleted: .SCAClient¶
import
org.oasisopen.sca.client.SCAClientFactory;

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

269 5 Error Handling

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

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

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

276 6 Asynchronous Programming

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

- 284 • support for non-blocking method calls
- 285 • callbacks

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

287 6.1 @OneWay

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

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

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

298 6.2 Callbacks

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

- 303 • an interface for the provided service
- 304 • a callback interface that is provided by the client

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

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

312 6.2.1 Using Callbacks

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

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

Deleted: March

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

```
324 package somepackage;  
325 import org.osoa.sca.annotation.Callback;  
326 import org.osoa.sca.annotation.Remotable;  
327 @Remotable  
328 @Callback(QuotationCallback.class)  
329 public interface Quotation {  
330     double requestQuotation(String productCode, int quantity);  
331 }  
332  
333 @Remotable  
334 public interface QuotationCallback {  
335     String getState();  
336     String getZipCode();  
337     String getCreditRating();  
338 }  
339
```

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

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

```
349  
350 @Callback  
351 protected QuotationCallback callback;  
352  
353 public double requestQuotation(String productCode, int quantity) {  
354     double price = getPrice(productCode, quantity);  
355     double discount = 0;  
356     if (quantity > 1000 && callback.getState().equals("FL")) {  
357         discount = 0.05;  
358     }  
359     if (quantity > 10000 && callback.getCreditRating().charAt(0) == 'A') {  
360         discount += 0.05;  
361     }  
362     return price * (1-discount);  
363 }  
364
```

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

```
368  
369 public class ClientImpl implements ClientService, QuotationCallback {  
370  
371     private QuotationService myService;  
372  
373     @Reference  
374     public void setMyService(QuotationService service) {  
375         myService = service;  
376     }  
377 }  
378
```

Deleted: March

```

376     }
377
378     public void aClientMethod() {
379         ...
380         double quote = myService.requestQuotation("AB123", 2000);
381         ...
382     }
383
384     public String getState() {
385         return "TX";
386     }
387     public String getZipCode() {
388         return "78746";
389     }
390     public String getCreditRating() {
391         return "AA";
392     }
393 }

```

394
395 In this example the callback is **stateless**, i.e., the callback requests do not need any information
396 relating to the original service request. For a callback that needs information relating to the
397 original service request (a **stateful** callback), this information can be passed to the client by the
398 service provider as parameters on the callback request.

399 6.2.2 Callback Instance Management

400 Instance management for callback requests received by the client of the bidirectional service is
401 handled in the same way as instance management for regular service requests. If the client
402 implementation has STATELESS scope, the callback is dispatched using a newly initialized
403 instance. If the client implementation has COMPOSITE scope, the callback is dispatched using the
404 same shared instance that is used to dispatch regular service requests.

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

411 6.2.3 Implementing Multiple Bidirectional Interfaces

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

```

418 @Callback
419 protected MyService1Callback callback1;
420
421 @Callback
422 protected MyService2Callback callback2;

```

424
425 If a single callback has a type that is compatible with multiple declared callback fields, then all of
426 them will be set.

Deleted: March

427 6.2.4 Accessing Callbacks

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

431
432 A reference implementing the callback service interface can be obtained using
433 `ServiceReference.getService()`.

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

```
435 @Callback
436 protected ServiceReference<MyCallback> callback;
437
438 public void someMethod() {
439     MyCallback myCallback = callback.getCallback();    ...
440
441     myCallback.receiveResult(theResult);
442 }
443
444 }
```

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

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

```
452 public void someMethod() {
453     MyCallback myCallback =
454         ComponentContext.getRequestContext().getCallback();
455     ...
456     ...
457     myCallback.receiveResult(theResult);
458 }
459
460 }
```

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

```
464 @Context
465 protected RequestContext requestContext;
466
467 void receiveResult(Object theResult) {
468     Object refParams =
469         requestContext.getServiceReference().getCallbackID();
470     ...
471 }
472 }
```

473 This is necessary if the service implementation has COMPOSITE scope, because callback injection
474 is not performed for composite-scoped implementations.
475

476 7 Policy Annotations for Java

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

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

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

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

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

503 7.1 General Intent Annotations

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

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

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

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

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

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

```
516 public static final String SCA_PREFIX=  
517     "{http://docs.oasis-open.org/ns/opencsa/sca/200712}";  
518 public static final String CONFIDENTIALITY = SCA_PREFIX + "confidentiality";  
519 public static final String CONFIDENTIALITY_MESSAGE = CONFIDENTIALITY + ".message";  
520
```

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

Deleted: March

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

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

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

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

529

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

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

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

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

```
549     package com.foo;
550
551     import static org.oasisopen.sca.Constants.*;
552     import static org.oasisopen.sca.annotation.Reference;
553     import static org.oasisopen.sca.annotation.Requires;
554
555     public class Foo {
556         @Requires(SCA_PREFIX+"confidentiality")
557         @Reference
558         public void setBar(Bar bar) {
559             ...
560         }
561     }
562
```

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

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

565 where

```
566     QualifiedIntent ::= QName('.' Qualifier)*
567     Qualifier ::= NCName
```

568

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

570 **7.2 Specific Intent Annotations**

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

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

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

582 @Integrity

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

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

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

588 The general form of specific intent annotations is:

589 '@' Intent ('(' qualifiers ')')?

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

591 Intent ::= NCName
592 qualifiers ::= "" qualifier "" (',' qualifier "")*
593 qualifier ::= NCName ('.' qualifier)?
594

595 **7.2.1 How to Create Specific Intent Annotations**

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

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

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

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

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

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

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

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

615 implies that both of the qualified intents "confidentiality.message" and "confidentiality.transport"
616 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

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

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

620 7.3 Application of Intent Annotations

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

- 622 • Java class
- 623 • Java interface
- 624 • Method
- 625 • Field
- 626 • Constructor parameter

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

```
629 @Authentication  
630 @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE})
```

631 In this case, the effective intents are "authentication", "confidentiality.message" and
632 "integrity.message".

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

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

639 7.3.1 Inheritance And Annotation

640 The inheritance rules for annotations are consistent with the common annotation specification, JSR
641 250 [JSR-250]

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

```
644 package services.hello;  
645 import org.oasisopen.sca.annotation.Remotable;  
646 import org.oasisopen.sca.annotation.Integrity;  
647 import org.oasisopen.sca.annotation.Authentication;  
648  
649 @Integrity("transport")  
650 @Authentication  
651 public class HelloService {  
652     @Integrity  
653     @Authentication("message")  
654     public String hello(String message) {...}  
655  
656     @Integrity  
657     @Authentication("transport")  
658     public String helloThere() {...}  
659 }  
660  
661 package services.hello;  
662 import org.oasisopen.sca.annotation.Remotable;  
663 import org.oasisopen.sca.annotation.Confidentiality;  
664 import org.oasisopen.sca.annotation.Authentication;
```



```

665
666     @Confidentiality("message")
667     public class HelloChildService extends HelloService {
668         @Confidentiality("transport")
669         public String hello(String message) {...}
670         @Authentication
671         String helloWorld() {...}
672     }

```

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

674

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

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

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

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

683

684 The listing below contains the equivalent declarative security interaction policy of the HelloService
685 and HelloChildService implementation corresponding to the Java interfaces and classes shown in
686 Example 2a.

687

```

688 <?xml version="1.0" encoding="ASCII"?>
689 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
690           name="HelloServiceComposite" >
691     <service name="HelloService" requires="integrity/transport
692           authentication">
693       ...
694     </service>
695     <service name="HelloChildService" requires="integrity/transport
696           authentication confidentiality/message">
697       ...
698     </service>
699     ...
700
701     <component name="HelloServiceComponent">*
702       <implementation.java class="services.hello.HelloService"/>
703       <operation name="hello" requires="integrity
704             authentication/message"/>
705       <operation name="helloThere"
706             requires="integrity
707             authentication/transport"/>
708     </component>
709     <component name="HelloChildServiceComponent">*
710       <implementation.java
711             class="services.hello.HelloChildService" />
712       <operation name="hello"
713             requires="confidentiality/transport"/>
714       <operation name="helloThere" requires=" integrity/transport
715             authentication"/>
716       <operation name="helloWorld" requires="authentication"/>
717     </component>
718

```

718

Deleted: March

719 ...
720
721 </composite>

722
723 Example 2b. Declaratives intents equivalent to annotated intents in Example 2a.

724

725 7.4 Relationship of Declarative And Annotated Intents

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

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

733 7.5 Policy Set Annotations

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

737
738 Policy Sets can be applied directly to Java implementations using the **@PolicySets** annotation.
739 The @PolicySets annotation either takes the QName of a single policy set as a string or the name
740 of two or more policy sets as an array of strings:

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

742

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

744 An example of the @PolicySets annotation:

745

```
746                   @Reference(name="helloService", required=true)  
747                   @PolicySets({ MY_NS + "WS_Encryption_Policy",  
748                                MY_NS + "WS_Authentication_Policy" })  
749                   public setHelloService(HelloService service) {  
750                    ...  
751                   }  
752
```

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

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

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

- 758 • Java class
- 759 • Java interface
- 760 • Method
- 761 • Field
- 762 • Constructor parameter

763 7.6 Security Policy Annotations

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

766 7.6.1 Security Interaction Policy

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

- 769 • @Integrity
- 770 • @Confidentiality
- 771 • @Authentication

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

- 773 • message
- 774 • transport

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

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

```
780
781 package services.hello;
782 //Interface for HelloService
783 public interface HelloService {
784     String hello(String helloMsg);
785 }
786
787 package services.client;
788 // Interface for ClientService
789 public interface ClientService {
790     public void clientMethod();
791 }
792
793 // Implementation class for ClientService
794 package services.client;
795
796 import services.hello.HelloService;
797 import org.oasisopen.sca.annotation.*;
798
799 @Service(ClientService.class)
800 public class ClientServiceImpl implements ClientService {
801
802     private HelloService helloService;
803
804     @Reference(name="helloService", required=true)
805     @Integrity("message")
806     @Authentication("message")
807     public void setHelloService(HelloService service) {
808         helloService = service;
809     }
810
811     public void clientMethod() {
812         String result = helloService.hello("Hello World!");
```

Deleted: March

```
813     ...
814     }
815 }
```

817 Example 1. Usage of annotated intents on a reference.

818 7.6.2 Security Implementation Policy

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

- 822 • RunAs

823 Takes as a parameter a string which is the name of a Security role.
824 eg. @RunAs("Manager") Code marked with this annotation executes with the Security
825 permissions of the identified role.
826

- 827 • RolesAllowed

828 Takes as a parameter a single string or an array of strings which represent one or more
829 role names. When present, the implementation can only be accessed by principals whose
830 role corresponds to one of the role names listed in the @roles attribute. How role names
831 are mapped to security principals is implementation dependent (SCA does not define this).
832 eg. @RolesAllowed({"Manager", "Employee"})
833

- 834 • PermitAll

835 No parameters. When present, grants access to all roles.
836

- 837 • DenyAll

838 No parameters. When present, denies access to all roles.
839

- 840 • DeclareRoles

841 Takes as a parameter a string or an array of strings which identify one or more role names
842 that form the set of roles used by the implementation.
843 eg. @DeclareRoles({"Manager", "Employee", "Customer"})

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

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

846 7.6.2.1 Annotated Implementation Policy Example

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

```
848
849 package services.account;
850 @Remotable
851 public interface AccountService {
852     AccountReport getAccountReport(String customerID);
853     float fromUSDollarToCurrency(float value);
854 }
```

855
856 The following is a full listing of the AccountServiceImpl class, showing the Service it implements,
857 plus the service references it makes and the settable properties that it has, along with a set of
858 implementation policy annotations:

```
859
860 package services.account;
```

```

861     import java.util.List;
862     import commonj.sdo.DataFactory;
863     import org.oasisopen.sca.annotation.Property;
864     import org.oasisopen.sca.annotation.Reference;
865     import org.oasisopen.sca.annotation.RolesAllowed;
866     import org.oasisopen.sca.annotation.RunAs;
867     import org.oasisopen.sca.annotation.PermitAll;
868     import services.accountdata.AccountDataService;
869     import services.accountdata.CheckingAccount;
870     import services.accountdata.SavingsAccount;
871     import services.accountdata.StockAccount;
872     import services.stockquote.StockQuoteService;
873     @RolesAllowed("customers")
874     @RunAs("accountants" )
875     public class AccountServiceImpl implements AccountService {
876
877         @Property
878         protected String currency = "USD";
879
880         @Reference
881         protected AccountDataService accountDataService;
882         @Reference
883         protected StockQuoteService stockQuoteService;
884
885         @RolesAllowed({"customers", "accountants"})
886         public AccountReport getAccountReport(String customerID) {
887
888             DataFactory dataFactory = DataFactory.INSTANCE;
889             AccountReport accountReport =
890                 (AccountReport)dataFactory.create(AccountReport.class);
891             List accountSummaries = accountReport.getAccountSummaries();
892
893             CheckingAccount checkingAccount =
894                 accountDataService.getCheckingAccount(customerID);
895             AccountSummary checkingAccountSummary =
896                 (AccountSummary)dataFactory.create(AccountSummary.class);
897
898             checkingAccountSummary.setAccountNumber(checkingAccount.getAccountNumber()
899 );
900             checkingAccountSummary.setAccountType("checking");
901             checkingAccountSummary.setBalance(fromUSDollarToCurrency
902                 (checkingAccount.getBalance()));
903             accountSummaries.add(checkingAccountSummary);
904
905             SavingsAccount savingsAccount =
906                 accountDataService.getSavingsAccount(customerID);
907             AccountSummary savingsAccountSummary =
908                 (AccountSummary)dataFactory.create(AccountSummary.class);
909
910             savingsAccountSummary.setAccountNumber(savingsAccount.getAccountNumber());
911             savingsAccountSummary.setAccountType("savings");
912             savingsAccountSummary.setBalance(fromUSDollarToCurrency
913                 (savingsAccount.getBalance()));
914             accountSummaries.add(savingsAccountSummary);
915
916             StockAccount stockAccount =
917                 accountDataService.getStockAccount(customerID);
918             AccountSummary stockAccountSummary =

```

Deleted: March

```

919         (AccountSummary) dataFactory.create(AccountSummary.class);
920     stockAccountSummary.setAccountNumber(stockAccount.getAccountNumber());
921     stockAccountSummary.setAccountType("stock");
922     float balance= (stockQuoteService.getQuote(stockAccount.getSymbol()))*
923         stockAccount.getQuantity();
924     stockAccountSummary.setBalance(fromUSDollarToCurrency(balance));
925     accountSummaries.add(stockAccountSummary);
926
927     return accountReport;
928 }
929
930 @PermitAll
931 public float fromUSDollarToCurrency(float value) {
932
933     if (currency.equals("USD")) return value;
934     if (currency.equals("EURO")) return value * 0.8f;
935     return 0.0f;
936 }
937 }

```

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

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

- 940 • @RolesAllowed("customers") - indicating that customers have access to the
- 941 implementation as a whole
- 942 • @RunAs("accountants") - indicating that the code in the implementation runs with the
- 943 permissions of accountants

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

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

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

947 can be called by any role.

948 8 Java API

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

950 8.1 Component Context

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

```
952
953 package org.oasisopen.sca;
954
955 public interface ComponentContext {
956     String getURI();
957
958     <B> B getService(Class<B> businessInterface, String referenceName);
959
960     <B> ServiceReference<B> getServiceReference(Class<B> businessInterface,
961                                             String referenceName);
962
963     <B> Collection<B> getServices(Class<B> businessInterface,
964                               String referenceName);
965
966     <B> Collection<ServiceReference<B>> getServiceReferences(Class<B>
967                                                         businessInterface, String referenceName);
968
969     <B> ServiceReference<B> createSelfReference(Class<B>
970                                             businessInterface);
971
972     <B> ServiceReference<B> createSelfReference(Class<B> businessInterface,
973                                             String serviceName);
974
975     <B> B getProperty(Class<B> type, String propertyName);
976
977     <B, R extends ServiceReference<B>> R cast(B target)
978         throws IllegalArgumentException;
979
980     RequestContext getRequestContext();
981
982
983 }
```

- 984
- 985 • **getURI()** - returns the absolute URI of the component within the SCA domain
 - 986 • **getService(Class businessInterface, String referenceName)** - Returns a proxy for
987 the reference defined by the current component. The getService() method takes as its
988 input arguments the Java type used to represent the target service on the client and the
989 name of the service reference. It returns an object providing access to the service. The
990 returned object implements the Java interface the service is typed with.
991 **ComponentContext.getService method MUST throw an IllegalArgumentException if the**
992 **reference identified by the referenceName parameter has multiplicity of 0..n or**
993 **1..n.[JCA80001]**
 - 994 • **getServiceReference(Class businessInterface, String referenceName)** - Returns a
995 ServiceReference defined by the current component. This method MUST throw an
996 IllegalArgumentException if the reference has multiplicity greater than one.

Deleted: March

- 997 • **getServices(Class businessInterface, String referenceName)** – Returns a list of
998 typed service proxies for a business interface type and a reference name.
- 999 • **getServiceReferences(Class businessInterface, String referenceName)** –Returns a
1000 list typed service references for a business interface type and a reference name.
- 1001 • **createSelfReference(Class businessInterface)** – Returns a ServiceReference that can
1002 be used to invoke this component over the designated service.
- 1003 • **createSelfReference(Class businessInterface, String serviceName)** – Returns a
1004 ServiceReference that can be used to invoke this component over the designated service.
1005 Service name explicitly declares the service name to invoke
- 1006 • **getProperty (Class type, String propertyName)** - Returns the value of an SCA
1007 property defined by this component.
- 1008 • **getRequestContext()** - Returns the context for the current SCA service request, or null if
1009 there is no current request or if the context is unavailable. The
1010 **ComponentContext.getRequestContext** method MUST return non-null when invoked during
1011 the execution of a Java business method for a service operation or a callback operation, on
1012 the same thread that the SCA runtime provided, and MUST return null in all other cases.
1013 **[JCA80002]**
- 1014 • **cast(B target)** - Casts a type-safe reference to a ServiceReference

1015 A component can access its component context by defining a field or setter method typed by
1016 **org.oasisopen.sca.ComponentContext** and annotated with **@Context**. To access the target
1017 service, the component uses **ComponentContext.getService(..)**.

1018 The following shows an example of component context usage in a Java class using the @Context
1019 annotation.

```
1020 private ComponentContext componentContext;
1021
1022 @Context
1023 public void setContext(ComponentContext context) {
1024     componentContext = context;
1025 }
1026
1027 public void doSomething() {
1028     HelloWorld service =
1029     componentContext.getService(HelloWorld.class, "HelloWorldComponent");
1030     service.hello("hello");
1031 }
```

1032 8.2 Request Context

1033 The following shows the **RequestContext** interface:

```
1034 package org.oasisopen.sca;
1035
1036 import javax.security.auth.Subject;
1037
1038 public interface RequestContext {
1039
1040     Subject getSecuritySubject();
1041
1042     String getServiceName();
1043     <CB> ServiceReference<CB> getCallbackReference();
1044     <CB> CB getCallback();
1045     <B> ServiceReference<B> getServiceReference();
1046 }
```

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

1047
1048 }
1049

1050 The RequestContext interface has the following methods:

- 1051 • **getSecuritySubject()** – Returns the JAAS Subject of the current request
- 1052 • **getServiceName()** – Returns the name of the service on the Java implementation the
1053 request came in on
- 1054 • **getCallbackReference()** – Returns a service reference to the callback as specified by the
1055 caller. This method returns null when called for a service request whose interface is not
1056 bidirectional or when called for a callback request.
- 1057 • **getCallback()** – Returns a proxy for the callback as specified by the caller. Similar to the
1058 getCallbackReference() method, this method returns null when called for a service request
1059 whose interface is not bidirectional or when called for a callback request.
- 1060 • **getServiceReference()** – When invoked during the execution of a service operation, the
1061 getServiceReference method MUST return a ServiceReference that represents the service
1062 that was invoked. When invoked during the execution of a callback operation, the
1063 getServiceReference method MUST return a ServiceReference that represents the callback
1064 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.

1065 8.3 ServiceReference

1066 ServiceReferences can be injected using the @Reference annotation on a field, a setter method,
1067 or constructor parameter taking the type ServiceReference. The detailed description of the usage
1068 of these methods is described in the section on Asynchronous Programming in this document.

1069 The following Java code defines the **ServiceReference** interface:

```
1070 package org.oasisopen.sca;  
1071  
1072 public interface ServiceReference<B> extends java.io.Serializable {  
1073     B getService();  
1074     Class<B> getBusinessInterface();  
1075 }  
1076  
1077
```

1078 The ServiceReference interface has the following methods:

- 1079 • **getService()** - Returns a type-safe reference to the target of this reference. The instance
1080 returned is guaranteed to implement the business interface for this reference. The value
1081 returned is a proxy to the target that implements the business interface associated with this
1082 reference.
- 1083 • **getBusinessInterface()** – Returns the Java class for the business interface associated with
1084 this reference.

1085 8.4 ServiceRuntimeException

1086 The following snippet shows the **ServiceRuntimeException**.

```
1087  
1088 package org.oasisopen.sca;  
1089  
1090 public class ServiceRuntimeException extends RuntimeException {  
1091     ...  
1092 }  
1093  
1094
```

1094 This exception signals problems in the management of SCA component execution.

Deleted: March

1095 8.5 ServiceUnavailableException

1096 The following snippet shows the *ServiceUnavailableException*.

```
1097 package org.oasisopen.sca;
1098
1099 public class ServiceUnavailableException extends ServiceRuntimeException {
1100     ...
1101 }
1102
1103
```

1104 This exception signals problems in the interaction with remote services. These are exceptions
1105 that can be transient, so retrying is appropriate. Any exception that is a ServiceRuntimeException
1106 that is *not* a ServiceUnavailableException is unlikely to be resolved by retrying the operation, since
1107 it most likely requires human intervention

1108 8.6 InvalidServiceException

1109 The following snippet shows the *InvalidServiceException*.

```
1110 package org.oasisopen.sca;
1111
1112 public class InvalidServiceException extends ServiceRuntimeException {
1113     ...
1114 }
1115
1116
```

1117 This exception signals that the ServiceReference is no longer valid. This can happen when the
1118 target of the reference is undeployed. This exception is not transient and therefore is unlikely to
1119 be resolved by retrying the operation and will most likely require human intervention.

1120 8.7 Constants

1121 The SCA *Constants* interface defines a number of constant values that are used in the SCA Java
1122 APIs and Annotations. The following snippet shows the Constants interface:

```
1123 package org.oasisopen.sca;
1124
1125 public interface Constants {
1126     String SCA_NS="http://docs.oasis-open.org/ns/opencsa/sca/200712";
1127     String SCA_PREFIX = "{ "+SCA_NS+"}";
1128 }
1129
```

1130 8.8 SCAClientFactory Class

1131 [The SCAClientFactory class provides the means for client code to obtain a proxy reference object](#)
1132 [for a service within an SCA Domain, through which the client code can invoke operations of that](#)
1133 [service. This is particularly useful for client code that is running outside the SCA Domain](#)
1134 [containing the target service, for example where the code is "unmanaged" and is not running](#)
1135 [under an SCA runtime.](#)

1136 [The SCAClientFactory is an abstract class which provides a set of static newInstance\(...\) methods](#)
1137 [which the client can invoke in order to obtain a concrete object implementing the](#)
1138 [SCAClientFactory interface for a particular SCA Domain. The returned SCAClientFactory object](#)
1139 [provides a getService\(\) method which provides the client with the means to obtain a reference](#)
1140 [proxy object for a service running in the SCA Domain.](#)

1141 [The SCAClientFactory class is as follows:](#)

Deleted: ¶

Formatted: Snippet Char
Char Char, Indent: Before:
0.5", First line: 0", Pattern:
Clear (Auto)

Formatted: Bullets and
Numbering

Deleted: March

```

1142
1143 /*
1144  * Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
1145  * OASIS trademark, IPR and other policies apply.
1146  */
1147 package org.oasisopen.sca.client;
1148
1149 import java.net.URI;
1150 import java.util.Properties;
1151
1152 import org.oasisopen.sca.NoSuchDomainException;
1153 import org.oasisopen.sca.NoSuchServiceException;
1154 import org.oasisopen.sca.client.SCAClientFactoryFinder;
1155 import org.oasisopen.sca.client.impl.SCAClientFactoryFinderImpl;
1156
1157 /**
1158  * The SCAClientFactory can be used by non-SCA managed code to
1159  * lookup services that exist in a SCADomain.
1160  *
1161  * @see SCAClientFactoryFinderImpl
1162  * @see SCAClient
1163  *
1164  * @author OASIS Open
1165  */
1166 public abstract class SCAClientFactory {
1167
1168     protected static SCAClientFactoryFinder factoryFinder;
1169
1170     private SCAClientFactory() {}
1171
1172     protected SCAClientFactory(URI domainURI) {...}
1173
1174     public URI getDomainURI() {...}
1175
1176     public static SCAClientFactory newInstance( URI domainURI )
1177         throws NoSuchDomainException {...}
1178
1179     public static SCAClientFactory newInstance(Properties properties,
1180         URI domainURI)
1181         throws NoSuchDomainException {...}
1182
1183     public static SCAClientFactory newInstance(ClassLoader classLoader,
1184         URI domainURI)
1185         throws NoSuchDomainException {...}
1186
1187     public static SCAClientFactory newInstance(Properties properties,
1188         ClassLoader classLoader,
1189         URI domainURI)
1190         throws NoSuchDomainException {...}
1191
1192     public abstract <T> T getService(Class<T> interfaze, String serviceURI)
1193         throws NoSuchServiceException, NoSuchDomainException;
1194 }
1195
1196 -----
1197 newInstance ( URI ) method:
1198 Obtains a object implementing the SCAClient interface.

```

Deleted: ¶

Deleted: March

1199 [Returns:](#)

1200

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

1201 [Parameters:](#)

1202

- [domainURI](#) - a URI for the SCA Domain which is targeted by the returned SCAClient

1203 [object](#)

1204 [Exceptions:](#)

1205

- [NoSuchDomainException](#) - thrown if the domainURI parameter does not identify a valid

1206 [SCA Domain](#)

1207 **[newInstance\(Properties, URI\) method:](#)**

1208 [Obtains a object implementing the SCAClient interface, using a specified set of properties.](#)

1209 [Returns:](#)

1210

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

1211 [Parameters:](#)

1212

- [properties](#) - a set of Properties that can be used when creating the object which

1213 [implements the SCAClient interface.](#)

1214

- [domainURI](#) - a URI for the SCA Domain which is targeted by the returned SCAClient

1215 [object](#)

1216 [Exceptions:](#)

1217

- [NoSuchDomainException](#) - thrown if the domainURI parameter does not identify a valid

1218 [SCA Domain](#)

1219 **[newInstance\(Classloader, URI\) method:](#)**

1220 [Obtains a object implementing the SCAClient interface using a specified classloader.](#)

1221 [Returns:](#)

1222

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

1223 [Parameters:](#)

1224

- [classLoader](#) - a ClassLoader to use when creating the object which implements the

1225 [SCAClient interface.](#)

1226

- [domainURI](#) - a URI for the SCA Domain which is targeted by the returned SCAClient

1227 [object](#)

1228 [Exceptions:](#)

1229

- [NoSuchDomainException](#) - thrown if the domainURI parameter does not identify a valid

1230 [SCA Domain](#)

1231 **[newInstance\(Properties, Classloader, URI\) method:](#)**

1232 [Obtains a object implementing the SCAClient interface using a specified set of properties and a](#)

1233 [specified classloader.](#)

1234 [Returns:](#)

1235

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

1236 [Parameters:](#)

1237

- [properties](#) - a set of Properties that can be used when creating the object which

1238 [implements the SCAClient interface.](#)

1239

- [classLoader](#) - a ClassLoader to use when creating the object which implements the

1240 [SCAClient interface.](#)

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

Formatted: Bullets and Numbering

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

Formatted: Bullets and Numbering

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

Deleted: March

1241 • [domainURI](#) - a URI for the SCA Domain which is targeted by the returned SCAClient
1242 object

Formatted: Bullets and Numbering

1243 Exceptions:

1244 • [NoSuchDomainException](#) - thrown if the domainURI parameter does not identify a valid
1245 SCA Domain

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

1246 **getService method:**

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

1247 Obtains a proxy reference object for a specified target service in a specified SCA Domain.

1248 Returns:

1249 • [proxy object](#) which implements the business interface T
1250 Invocations of a business method of the proxy causes the invocation of the corresponding
1251 operation of the target service .

Formatted: Bullets and Numbering

1252 Parameters:

1253 • [interfaze](#) - a Java interface class which is the business interface of the target service
1254 • [serviceURI](#) - a String containing the relative URI of the target service within its SCA
1255 Domain.
1256 Takes the form [componentName/serviceName](#) or can also take the extended form
1257 [componentName/serviceName/bindingName](#) to use a specific binding of the target service
1258

Formatted: Bullets and Numbering

1259 Exceptions:

1260 • [NoSuchServiceException](#) - thrown if a service with the relative URI [serviceURI](#) and a
1261 business interface which matches [interfaze](#) cannot be found in the SCA Domain targeted
1262 by the SCAClient object

Formatted: Bullets and Numbering

1263 • [NoSuchDomainException](#) - thrown if the domainURI of the SCAClientFactory does not
1264 identify a valid SCA Domain

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

1265 **SCAClientFactory (URI) method:** a single argument constructor that must be available on all
1266 concrete subclasses of SCAClientFactory. The URI required is the URI of the Domain targeted by
1267 the SCAClientFactory

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

1268 **getDomainURI() method:**

1269 Obtains the Domain URI value for this SCAClientFactory

1270 Returns:

1271 • [URI](#) of the target SCA Domain for this SCAClientFactory

Formatted: Bullets and Numbering

1272 Parameters:

1273 • [none](#)

Formatted: Bullets and Numbering

1274 Exceptions:

1275 • [none](#)

Formatted: Bullets and Numbering

1276 **private SCAClientFactory() method:** this private no-argument constructor prevents
1277 instantiation of an SCAClientFactory instance without the use of the constructor with an argument,
1278 even by subclasses of the abstract SCAClientFactory class.

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

1279 **factoryFinder protected field:**

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

1280 Provides a means by which a provider of an SCAClientFactory implementation can inject a factory
1281 finder implementation into the abstract SCAClientFactory class - once this is done, future
1282 invocations of the SCAClientFactory use the injected factory finder to locate and return an instance
1283 of a subclass of SCAClientFactory.

Deleted: March

1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299

1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332

8.9 [SCAClientFactoryFinder Interface](#)

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

```
package org.oasisopen.sca.client;  
  
public interface SCAClientFactoryFinder {  
    SCAClientFactory find(Properties properties,  
                          ClassLoader classLoader,  
                          URI domainURI )  
    throws NoSuchDomainException ;  
}
```

Formatted: Bullets and Numbering

Formatted: Indent: Before: 0.25"

Formatted: Indent: Before: 0.25"

8.10 [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 NoSuchDomainException, ServiceRuntimeException {...}  
    ...  
}
```

Formatted: Bullets and Numbering

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

Formatted: Indent: First line: 0.25"

[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:

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Deleted: March

1333 • [SCAClientFactory](#) implementation object

1334 Parameters:

1335 • [properties](#) - a set of Properties that can be used when creating the object which
1336 implements the SCAClientFactory interface.

1337 • [classLoader](#) - a ClassLoader to use when creating the object which implements the
1338 SCAClientFactory interface.

1339 • [domainURI](#) - a URI for the SCA Domain targeted by the SCAClientFactory

1340 Exceptions:

1341 • [ServiceRuntimeException](#) - if the SCAClientFactory implementation could not be found

1342

1343 [8.11 NoSuchDomainException](#)

1344 The following shows the NoSuchDomainException:

1345 `package org.oasisopen.sca;`

1346 `public class NoSuchDomainException extends Exception {`
1347 `...
1348 }
1349`

1350 This exception indicates that the Domain specified could not be found.

1351 [8.12 NoSuchServiceException](#)

1352 The following shows the NoSuchServiceException:

1353 `package org.oasisopen.sca;`

1354 `public class NoSuchServiceException extends Exception {`
1355 `...
1356 }
1357`

1358 This exception indicates that the service specified could not be found.

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

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

1359 9 Java Annotations

1360 This section provides definitions of all the Java annotations which apply to SCA.

1361 This specification places constraints on some annotations that are not detectable by a Java
1362 compiler. For example, the definition of the @Property and @Reference annotations indicate that
1363 they are allowed on parameters, but sections 8.14 and 8.15 constrain those definitions to
1364 constructor parameters. An SCA runtime MUST verify the proper use of all SCA annotations and if
1365 an annotation is improperly used, the SCA runtime MUST NOT run the component which uses the
1366 invalid implementation code. [JCA90001]

1367 SCA annotations MUST NOT be used on static methods or on static fields. It is an error to use an
1368 SCA annotation on a static method or a static field of an implementation class and the SCA
1369 runtime MUST NOT instantiate such an implementation class. [JCA90002]

1370 9.1 @AllowsPassByReference

1371 The following Java code defines the **@AllowsPassByReference** annotation:

```
1372  
1373 package org.oasisopen.sca.annotation;  
1374  
1375 import static java.lang.annotation.ElementType.TYPE;  
1376 import static java.lang.annotation.ElementType.METHOD;  
1377 import static java.lang.annotation.RetentionPolicy.RUNTIME;  
1378 import java.lang.annotation.Retention;  
1379 import java.lang.annotation.Target;  
1380  
1381 @Target({TYPE, METHOD})  
1382 @Retention(RUNTIME)  
1383 public @interface AllowsPassByReference {  
1384  
1385 }  
1386
```

1387 The **@AllowsPassByReference** annotation is used on implementations of remotable interfaces to
1388 indicate that interactions with the service from a client within the same address space are allowed
1389 to use pass by reference data exchange semantics. The implementation promises that its by-value
1390 semantics will be maintained even if the parameters and return values are actually passed by-
1391 reference. This means that the service will not modify any operation input parameter or return
1392 value, even after returning from the operation. Either a whole class implementing a remotable
1393 service or an individual remotable service method implementation can be annotated using the
1394 @AllowsPassByReference annotation.

1395 @AllowsPassByReference has no attributes

1396 The following snippet shows a sample where @AllowsPassByReference is defined for the
1397 implementation of a service method on the Java component implementation class.

```
1398  
1399 @AllowsPassByReference  
1400 public String hello(String message) {  
1401     ...  
1402 }
```

1403 9.2 @Authentication

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


```

1405
1406 package org.oasisopen.sca.annotation;
1407
1408 import static java.lang.annotation.ElementType.FIELD;
1409 import static java.lang.annotation.ElementType.METHOD;
1410 import static java.lang.annotation.ElementType.PARAMETER;
1411 import static java.lang.annotation.ElementType.TYPE;
1412 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1413 import static org.oasisopen.sca.Constants.SCA_PREFIX;
1414
1415 import java.lang.annotation.Inherited;
1416 import java.lang.annotation.Retention;
1417 import java.lang.annotation.Target;
1418
1419 @Inherited
1420 @Target({TYPE, FIELD, METHOD, PARAMETER})
1421 @Retention(RUNTIME)
1422 @Intent(Authentication.AUTHENTICATION)
1423 public @interface Authentication {
1424     String AUTHENTICATION = SCA_PREFIX + "authentication";
1425     String AUTHENTICATION_MESSAGE = AUTHENTICATION + ".message";
1426     String AUTHENTICATION_TRANSPORT = AUTHENTICATION + ".transport";
1427
1428     /**
1429      * List of authentication qualifiers (such as "message"
1430      * or "transport").
1431      *
1432      * @return authentication qualifiers
1433      */
1434     @Qualifier
1435     String[] value() default "";
1436 }

```

1437 The **@Authentication** annotation is used to indicate that the invocation requires authentication.
1438 See the [section on Application of Intent Annotations](#) for samples and details.

1439 9.3 @Callback

1440 The following Java code defines the **@Callback** annotation:

```

1441
1442 package org.oasisopen.sca.annotation;
1443
1444 import static java.lang.annotation.ElementType.TYPE;
1445 import static java.lang.annotation.ElementType.METHOD;
1446 import static java.lang.annotation.ElementType.FIELD;
1447 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1448 import java.lang.annotation.Retention;
1449 import java.lang.annotation.Target;
1450
1451 @Target(TYPE, METHOD, FIELD)
1452 @Retention(RUNTIME)
1453 public @interface Callback {
1454
1455     Class<?> value() default Void.class;
1456 }
1457
1458

```

Deleted: March

1459 The @Callback annotation is used to annotate a service interface with a callback interface by
1460 specifying the Java class object of the callback interface as an attribute.

1461 The @Callback annotation has the following attribute:

- 1462 • **value** – the name of a Java class file containing the callback interface

1463

1464 The @Callback annotation can also be used to annotate a method or a field of an SCA
1465 implementation class, in order to have a callback object injected. When used to annotate a
1466 method or a field of an implementation class for injection of a callback object, the @Callback
1467 annotation MUST NOT specify any attributes. [JCA90046]

1468 An example use of the @Callback annotation to declare a callback interface follows:

```
1469 package somepackage;  
1470 import org.oasisopen.sca.annotation.Callback;  
1471 import org.oasisopen.sca.annotation.Remotable;  
1472 @Remotable  
1473 @Callback(MyServiceCallback.class)  
1474 public interface MyService {  
1475  
1476     void someMethod(String arg);  
1477 }  
1478  
1479 @Remotable  
1480 public interface MyServiceCallback {  
1481  
1482     void receiveResult(String result);  
1483 }  
1484
```

1485 In this example, the implied component type is:

```
1486 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" >  
1487     <service name="MyService">  
1488         <interface.java interface="somepackage.MyService"  
1489             callbackInterface="somepackage.MyServiceCallback"/>  
1490     </service>  
1491 </componentType>
```

1493 9.4 @ComponentName

1494 The following Java code defines the @ComponentName annotation:

1495

```
1496 package org.oasisopen.sca.annotation;  
1497  
1498 import static java.lang.annotation.ElementType.METHOD;  
1499 import static java.lang.annotation.ElementType.FIELD;  
1500 import static java.lang.annotation.RetentionPolicy.RUNTIME;  
1501 import java.lang.annotation.Retention;  
1502 import java.lang.annotation.Target;  
1503  
1504 @Target({METHOD, FIELD})  
1505 @Retention(RUNTIME)  
1506 public @interface ComponentName {  
1507  
1508 }  
1509
```

1510 The @ComponentName annotation is used to denote a Java class field or setter method that is
1511 used to inject the component name.

1512 The following snippet shows a component name field definition sample.

```
1513  
1514 @ComponentName  
1515 private String componentName;  
1516
```

1517 The following snippet shows a component name setter method sample.

```
1518  
1519 @ComponentName  
1520 public void setComponentName(String name) {  
1521     //...  
1522 }
```

1523 9.5 @Confidentiality

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

```
1525  
1526 package org.oasisopen.sca.annotations;  
1527  
1528 import static java.lang.annotation.ElementType.FIELD;  
1529 import static java.lang.annotation.ElementType.METHOD;  
1530 import static java.lang.annotation.ElementType.PARAMETER;  
1531 import static java.lang.annotation.ElementType.TYPE;  
1532 import static java.lang.annotation.RetentionPolicy.RUNTIME;  
1533 import static org.oasisopen.sca.Constants.SCA_PREFIX;  
1534  
1535 import java.lang.annotation.Inherited;  
1536 import java.lang.annotation.Retention;  
1537 import java.lang.annotation.Target;  
1538  
1539 @Inherited  
1540 @Target({TYPE, FIELD, METHOD, PARAMETER})  
1541 @Retention(RUNTIME)  
1542 @Intent(Confidentiality.CONFIDENTIALITY)  
1543 public @interface Confidentiality {  
1544     String CONFIDENTIALITY = SCA_PREFIX + "confidentiality";  
1545     String CONFIDENTIALITY_MESSAGE = CONFIDENTIALITY + ".message";  
1546     String CONFIDENTIALITY_TRANSPORT = CONFIDENTIALITY + ".transport";  
1547  
1548     /**  
1549      * List of confidentiality qualifiers such as "message" or  
1550      * "transport".  
1551      *  
1552      * @return confidentiality qualifiers  
1553      */  
1554     @Qualifier  
1555     String[] value() default "";  
1556 }
```

1557 The **@Confidentiality** annotation is used to indicate that the invocation requires confidentiality.

1558 See the [section on Application of Intent Annotations](#) for samples and details.

1559 9.6 @Constructor

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

```
1561 package org.oasisopen.sca.annotation;
1562
1563 import static java.lang.annotation.ElementType.CONSTRUCTOR;
1564 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1565 import java.lang.annotation.Retention;
1566 import java.lang.annotation.Target;
1567
1568 @Target({CONSTRUCTOR})
1569 @Retention(RUNTIME)
1570 public @interface Constructor { }
```

1573 The @Constructor annotation is used to mark a particular constructor to use when instantiating a
1574 Java component implementation. If a constructor of an implementation class is annotated with
1575 @Constructor and the constructor has parameters, each of these parameters MUST have either a
1576 @Property annotation or a @Reference annotation. [JCA90003]

1577 The following snippet shows a sample for the @Constructor annotation.

```
1578
1579 public class HelloServiceImpl implements HelloService {
1580
1581     public HelloServiceImpl(){
1582         ...
1583     }
1584
1585     @Constructor
1586     public HelloServiceImpl(@Property(name="someProperty")
1587                             String someProperty ){
1588         ...
1589     }
1590
1591     public String hello(String message) {
1592         ...
1593     }
1594 }
```

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

1595 9.7 @Context

1596 The following Java code defines the **@Context** annotation:

```
1597
1598 package org.oasisopen.sca.annotation;
1599
1600 import static java.lang.annotation.ElementType.METHOD;
1601 import static java.lang.annotation.ElementType.FIELD;
1602 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1603 import java.lang.annotation.Retention;
1604 import java.lang.annotation.Target;
1605
1606 @Target({METHOD, FIELD})
1607 @Retention(RUNTIME)
1608 public @interface Context { }
```

Deleted: March

1610 }
1611

1612 The @Context annotation is used to denote a Java class field or a setter method that is used to
1613 inject a composite context for the component. The type of context to be injected is defined by the
1614 type of the Java class field or type of the setter method input argument; the type is either
1615 **ComponentContext** or **RequestContext**.

1616 The @Context annotation has no attributes.

1617 The following snippet shows a ComponentContext field definition sample.

```
1618  
1619 @Context  
1620 protected ComponentContext context;  
1621
```

1622 The following snippet shows a RequestContext field definition sample.

```
1623  
1624 @Context  
1625 protected RequestContext context;
```

1626 9.8 @Destroy

1627 The following Java code defines the **@Destroy** annotation:

```
1628  
1629 package org.oasisopen.sca.annotation;  
1630  
1631 import static java.lang.annotation.ElementType.METHOD;  
1632 import static java.lang.annotation.RetentionPolicy.RUNTIME;  
1633 import java.lang.annotation.Retention;  
1634 import java.lang.annotation.Target;  
1635  
1636 @Target(METHOD)  
1637 @Retention(RUNTIME)  
1638 public @interface Destroy {  
1639  
1640 }  
1641
```

1642 The @Destroy annotation is used to denote a single Java class method that will be called when the
1643 scope defined for the implementation class ends. A method annotated with @Destroy MAY have
1644 any access modifier and MUST have a void return type and no arguments. [JCA90004]

1645 If there is a method annotated with @Destroy that matches the criteria for the annotation, the
1646 SCA runtime MUST call the annotated method when the scope defined for the implementation
1647 class ends. [JCA90005]

1648 The following snippet shows a sample for a destroy method definition.

```
1649  
1650 @Destroy  
1651 public void myDestroyMethod() {  
1652     ...  
1653 }
```

1654 9.9 @EagerInit

1655 The following Java code defines the **@EagerInit** annotation:

1656

```
1657 package org.oasisopen.sca.annotation;
1658
1659 import static java.lang.annotation.ElementType.TYPE;
1660 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1661 import java.lang.annotation.Retention;
1662 import java.lang.annotation.Target;
1663
1664 @Target(TYPE)
1665 @Retention(RUNTIME)
1666 public @interface EagerInit {
1667
1668 }
1669
1670 The @EagerInit annotation is used to mark the Java class of a COMPOSITE scoped
1671 implementation for eager initialization. When marked for eager initialization with an @EagerInit
1672 annotation, the composite scoped instance MUST be created when its containing component is
1673 started. [JCA90007]
```

1674 9.10 @Init

1675 The following Java code defines the **@Init** annotation:

```
1676
1677 package org.oasisopen.sca.annotation;
1678
1679 import static java.lang.annotation.ElementType.METHOD;
1680 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1681 import java.lang.annotation.Retention;
1682 import java.lang.annotation.Target;
1683
1684 @Target(METHOD)
1685 @Retention(RUNTIME)
1686 public @interface Init {
1687
1688 }
1689
1690
```

1691 The @Init annotation is used to denote a single Java class method that is called when the scope
1692 defined for the implementation class starts. A method marked with the @Init annotation MAY
1693 have any access modifier and MUST have a void return type and no arguments. [JCA90008]

1694 If there is a method annotated with @Init that matches the criteria for the annotation, the SCA
1695 runtime MUST call the annotated method after all property and reference injection is complete.
1696 [JCA90009]

1697 The following snippet shows an example of an init method definition.

```
1698
1699 @Init
1700 public void myInitMethod() {
1701     ...
1702 }
```

1703 9.11 @Integrity

1704 The following Java code defines the **@Integrity** annotation:

1705

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

```

1706 package org.oasisopen.sca.annotation;
1707
1708 import static java.lang.annotation.ElementType.FIELD;
1709 import static java.lang.annotation.ElementType.METHOD;
1710 import static java.lang.annotation.ElementType.PARAMETER;
1711 import static java.lang.annotation.ElementType.TYPE;
1712 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1713 import static org.oasisopen.Constants.SCA_PREFIX;
1714
1715 import java.lang.annotation.Inherited;
1716 import java.lang.annotation.Retention;
1717 import java.lang.annotation.Target;
1718
1719 @Inherited
1720 @Target({TYPE, FIELD, METHOD, PARAMETER})
1721 @Retention(RUNTIME)
1722 @Intent(Integrity.INTEGRITY)
1723 public @interface Integrity {
1724     String INTEGRITY = SCA_PREFIX + "integrity";
1725     String INTEGRITY_MESSAGE = INTEGRITY + ".message";
1726     String INTEGRITY_TRANSPORT = INTEGRITY + ".transport";
1727
1728     /**
1729      * List of integrity qualifiers (such as "message" or "transport").
1730      *
1731      * @return integrity qualifiers
1732      */
1733     @Qualifier
1734     String[] value() default "";
1735 }

```

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

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

1740 9.12 @Intent

1741 The following Java code defines the **@Intent** annotation:

```

1742 package org.oasisopen.sca.annotation;
1743
1744 import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
1745 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1746 import java.lang.annotation.Retention;
1747 import java.lang.annotation.Target;
1748
1749 @Target({ANNOTATION_TYPE})
1750 @Retention(RUNTIME)
1751 public @interface Intent {
1752     /**
1753      * The qualified name of the intent, in the form defined by
1754      * {@link javax.xml.namespace.QName#toString}.
1755      * @return the qualified name of the intent
1756      */
1757     String value() default "";
1758
1759     /**
1760

```

Deleted: March

```

1761     * The XML namespace for the intent.
1762     * @return the XML namespace for the intent
1763     */
1764     String targetNamespace() default "";
1765
1766     /**
1767     * The name of the intent within its namespace.
1768     * @return name of the intent within its namespace
1769     */
1770     String localPart() default "";
1771 }
1772

```

1773 The @Intent annotation is used for the creation of new annotations for specific intents. It is not
 1774 expected that the @Intent annotation will be used in application code.

1775 See the [section "How to Create Specific Intent Annotations"](#) for details and samples of how to
 1776 define new intent annotations.

1777 9.13 @OneWay

1778 The following Java code defines the **@OneWay** annotation:

```

1779
1780 package org.oasisopen.sca.annotation;
1781
1782 import static java.lang.annotation.ElementType.METHOD;
1783 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1784 import java.lang.annotation.Retention;
1785 import java.lang.annotation.Target;
1786
1787 @Target(METHOD)
1788 @Retention(RUNTIME)
1789 public @interface OneWay {
1790
1791 }
1792
1793

```

1794 The @OneWay annotation is used on a Java interface or class method to indicate that invocations
 1795 will be dispatched in a non-blocking fashion as described in the section on Asynchronous
 1796 [Programming](#).

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

1797 The @OneWay annotation has no attributes.

1798 The following snippet shows the use of the @OneWay annotation on an interface.

```

1799 package services.hello;
1800
1801 import org.oasisopen.sca.annotation.OneWay;
1802
1803 public interface HelloService {
1804     @OneWay
1805     void hello(String name);
1806 }

```

1807 9.14 @PolicySets

1808 The following Java code defines the **@PolicySets** annotation:

```

1809 package org.oasisopen.sca.annotation;
1810

```

Deleted: March


```

1811
1812 import static java.lang.annotation.ElementType.FIELD;
1813 import static java.lang.annotation.ElementType.METHOD;
1814 import static java.lang.annotation.ElementType.PARAMETER;
1815 import static java.lang.annotation.ElementType.TYPE;
1816 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1817
1818 import java.lang.annotation.Retention;
1819 import java.lang.annotation.Target;
1820
1821 @Target({TYPE, FIELD, METHOD, PARAMETER})
1822 @Retention(RUNTIME)
1823 public @interface PolicySets {
1824     /**
1825      * Returns the policy sets to be applied.
1826      *
1827      * @return the policy sets to be applied
1828      */
1829     String[] value() default "";
1830 }
1831

```

1832 The **@PolicySets** annotation is used to attach one or more SCA Policy Sets to a Java
1833 implementation class or to one of its subelements.

1834 See the [section "Policy Set Annotations"](#) for details and samples.

1835 9.15 @Property

1836 The following Java code defines the **@Property** annotation:

```

1837 package org.oasisopen.sca.annotation;
1838
1839 import static java.lang.annotation.ElementType.METHOD;
1840 import static java.lang.annotation.ElementType.FIELD;
1841 import static java.lang.annotation.ElementType.PARAMETER;
1842 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1843 import java.lang.annotation.Retention;
1844 import java.lang.annotation.Target;
1845
1846 @Target({METHOD, FIELD, PARAMETER})
1847 @Retention(RUNTIME)
1848 public @interface Property {
1849
1850     String name() default "";
1851     boolean required() default true;
1852 }
1853

```

1854 The @Property annotation is used to denote a Java class field, a setter method, or a constructor
1855 parameter that is used to inject an SCA property value. The type of the property injected, which
1856 can be a simple Java type or a complex Java type, is defined by the type of the Java class field or
1857 the type of the input parameter of the setter method or constructor.

1858 The @Property annotation can be used on fields, on setter methods or on a constructor method
1859 parameter. However, **the @Property annotation MUST NOT be used on a class field that is declared
1860 as final. [JCA90011]**

1861 Properties can also be injected via setter methods even when the @Property annotation is not
1862 present. However, **the @Property annotation MUST be used in order to inject a property onto a
1863 non-public field. [JCA90012]** In the case where there is no @Property annotation, the name of the
1864 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

1865 Where there is both a setter method and a field for a property, the setter method is used.

1866 The @Property annotation has the following attributes:

- 1867 • **name (optional)** – the name of the property. For a field annotation, the default is the
1868 name of the field of the Java class. For a setter method annotation, the default is the
1869 JavaBeans property name [JAVABEANS] corresponding to the setter method name. For a
1870 @Property annotation applied to a constructor parameter, there is no default value for the
1871 name attribute and the name attribute MUST be present. [JCA90013]
- 1872 • **required (optional)** – a boolean value which specifies whether injection of the property
1873 value is required or not, where true means injection is required and false means injection
1874 is not required. Defaults to true. For a @Property annotation applied to a constructor
1875 parameter, the required attribute MUST have the value true. [JCA90014]

1876

1877 The following snippet shows a property field definition sample.

1878

```
1879 @Property(name="currency", required=true)  
1880 protected String currency;
```

1881

1882 The following snippet shows a property setter sample

1883

```
1884 @Property(name="currency", required=true)  
1885 public void setCurrency( String theCurrency ) {  
1886     ....  
1887 }
```

1888

1889 For a @Property annotation, if the the type of the Java class field or the type of the input
1890 parameter of the setter method or constructor is defined as an array or as any type that extends
1891 or implements java.util.Collection, then the SCA runtime MUST introspect the component type of
1892 the implementation with a <property/> element with a @many attribute set to true, otherwise
1893 @many MUST be set to false. [JCA90047]

1894 The following snippet shows the definition of a configuration property using the @Property
1895 annotation for a collection.

```
1896 ...  
1897 private List<String> helloConfigurationProperty;  
1898  
1899 @Property(required=true)  
1900 public void setHelloConfigurationProperty(List<String> property) {  
1901     helloConfigurationProperty = property;  
1902 }  
1903 ...
```

1904 9.16 @Qualifier

1905 The following Java code defines the @Qualifier annotation:

```
1906 package org.oasisopen.sca.annotation;  
1907  
1908 import static java.lang.annotation.ElementType.METHOD;  
1909 import static java.lang.annotation.RetentionPolicy.RUNTIME;  
1910  
1911
```

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

```

1912 import java.lang.annotation.Retention;
1913 import java.lang.annotation.Target;
1914
1915 @Target(METHOD)
1916 @Retention(RUNTIME)
1917 public @interface Qualifier {
1918 }
1919

```

1920 The @Qualifier annotation is applied to an attribute of a specific intent annotation definition,
 1921 defined using the @Intent annotation, to indicate that the attribute provides qualifiers for the
 1922 intent. The @Qualifier annotation MUST be used in a specific intent annotation definition where the
 1923 intent has qualifiers. [JCA90015]

1924 See the section "How to Create Specific Intent Annotations" for details and samples of how to
 1925 define new intent annotations.

1926 9.17 @Reference

1927 The following Java code defines the @Reference annotation:

```

1928
1929 package org.oasisopen.sca.annotation;
1930
1931 import static java.lang.annotation.ElementType.METHOD;
1932 import static java.lang.annotation.ElementType.FIELD;
1933 import static java.lang.annotation.ElementType.PARAMETER;
1934 import static java.lang.annotation.RetentionPolicy.RUNTIME;
1935 import java.lang.annotation.Retention;
1936 import java.lang.annotation.Target;
1937 @Target({METHOD, FIELD, PARAMETER})
1938 @Retention(RUNTIME)
1939 public @interface Reference {
1940
1941     String name() default "";
1942     boolean required() default true;
1943 }
1944

```

1945 The @Reference annotation type is used to annotate a Java class field, a setter method, or a
 1946 constructor parameter that is used to inject a service that resolves the reference. The interface of
 1947 the service injected is defined by the type of the Java class field or the type of the input parameter
 1948 of the setter method or constructor.

1949 The @Reference annotation MUST NOT be used on a class field that is declared as final.
 1950 [JCA90016]

1951 References can also be injected via setter methods even when the @Reference annotation is not
 1952 present. However, the @Reference annotation MUST be used in order to inject a reference onto a
 1953 non-public field. [JCA90017] In the case where there is no @Reference annotation, the name of
 1954 the reference is the same as the name of the field or setter.

1955 Where there is both a setter method and a field for a reference, the setter method is used.

1956 The @Reference annotation has the following attributes:

- 1957 • **name : String (optional)** – the name of the reference. For a field annotation, the default is
 1958 the name of the field of the Java class. For a setter method annotation, the default is the
 1959 JavaBeans property name corresponding to the setter method name. For a @Reference
 1960 annotation applied to a constructor parameter, there is no default for the name attribute
 1961 and the name attribute MUST be present. [JCA90018]

- 1962
- 1963
- 1964
- 1965
- **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]

1966

1967 The following snippet shows a reference field definition sample.

1968

```
1969 @Reference(name="stockQuote", required=true)
1970 protected StockQuoteService stockQuote;
```

1971

1972 The following snippet shows a reference setter sample

1973

```
1974 @Reference(name="stockQuote", required=true)
1975 public void setStockQuote( StockQuoteService theSQService ) {
1976     ...
1977 }
```

1978

1979 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.

1983

```
1984 package services.hello;
1985
1986 private HelloService helloService;
1987
1988 @Reference(name="helloService", required=true)
1989 public setHelloService(HelloService service) {
1990     helloService = service;
1991 }
1992
1993 public void clientMethod() {
1994     String result = helloService.hello("Hello World!");
1995     ...
1996 }
1997
```

1998 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.

2001

```
2002 <?xml version="1.0" encoding="ASCII"?>
2003 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712">
2004
2005     <!-- Any services offered by the component would be listed here -->
2006     <reference name="helloService" multiplicity="1..1">
2007         <interface.java interface="services.hello.HelloService"/>
2008     </reference>
2009
2010 </componentType>
2011
```

Deleted: March

2012 If the type of a reference is not an array or any type that extends or implements
2013 java.util.Collection, then the SCA runtime MUST introspect the component type of the
2014 implementation with a <reference/> element with @multiplicity= 0..1 if the @Reference
2015 annotation required attribute is false and with @multiplicity=1..1 if the @Reference annotation
2016 required attribute is true, [JCA90020]

2017 If the type of a reference is defined as an array or as any type that extends or implements
2018 java.util.Collection, then the SCA runtime MUST introspect the component type of the
2019 implementation with a <reference/> element with @multiplicity=0..n if the @Reference
2020 annotation required attribute is false and with @multiplicity=1..n if the @Reference annotation
2021 required attribute is true, [JCA90021]

2022 The following fragment from a component implementation shows a sample of a service reference
2023 definition using the @Reference annotation on a java.util.List. The name of the reference is
2024 "helloServices" and its type is HelloService. The clientMethod() calls the "hello" operation of all the
2025 services referenced by the helloServices reference. In this case, at least one HelloService needs
2026 to be present, so **required** is true.

```
2027 @Reference(name="helloServices", required=true)  
2028 protected List<HelloService> helloServices;  
2029  
2030 public void clientMethod() {  
2031  
2032     ...  
2033     for (int index = 0; index < helloServices.size(); index++) {  
2034         HelloService helloService =  
2035             (HelloService)helloServices.get(index);  
2036         String result = helloService.hello("Hello World!");  
2037     }  
2038     ...  
2039 }  
2040  
2041 }
```

2042 The following snippet shows the XML representation of the component type reflected from for the
2043 former component implementation fragment. There is no need to author this component type in
2044 this case since it can be reflected from the Java class.

```
2045  
2046 <?xml version="1.0" encoding="ASCII"?>  
2047 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712">  
2048  
2049     <!-- Any services offered by the component would be listed here -->  
2050     <reference name="helloServices" multiplicity="1..n">  
2051         <interface.java interface="services.hello.HelloService"/>  
2052     </reference>  
2053  
2054 </componentType>
```

2055
2056 An unwired reference with a multiplicity of 0..1 MUST be presented to the implementation code by
2057 the SCA runtime as null, [JCA90022] An unwired reference with a multiplicity of 0..n MUST be
2058 presented to the implementation code by the SCA runtime as an empty array or empty collection,
2059 [JCA90023]

2060 9.17.1 Reinjection

2061 References MAY be reinjected by an SCA runtime after the initial creation of a component if the
2062 reference target changes due to a change in wiring that has occurred since the component was
2063 initialized. [JCA90024]

2064 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

2065 1. The component MUST NOT be STATELESS scoped.

2066 2. The reference MUST use either field-based injection or setter injection. References that are

2067 injected through constructor injection MUST NOT be changed.

2068 [JCA90025]

2069 Setter injection allows for code in the setter method to perform processing in reaction to a change.

2070 If a reference target changes and the reference is not reinjected, the reference MUST continue to

2071 work as if the reference target was not changed. [JCA90026]

2072 If an operation is called on a reference where the target of that reference has been undeployed,

2073 the SCA runtime SHOULD throw an InvalidServiceException. [JCA90027] If an operation is called

2074 on a reference where the target of the reference has become unavailable for some reason, the

2075 SCA runtime SHOULD throw a ServiceUnavailableException. [JCA90028] If the target service of

2076 the reference is changed, the reference MAY continue to work, depending on the runtime and the

2077 type of change that was made. [JCA90029] If it doesn't work, the exception thrown will depend on

2078 the runtime and the cause of the failure.

2079 A ServiceReference that has been obtained from a reference by ComponentContext.cast()

2080 corresponds to the reference that is passed as a parameter to cast(). If the reference is

2081 subsequently reinjected, the ServiceReference obtained from the original reference MUST continue

2082 to work as if the reference target was not changed. [JCA90030] If the target of a ServiceReference

2083 has been undeployed, the SCA runtime SHOULD throw a InvalidServiceException when an

2084 operation is invoked on the ServiceReference. [JCA90031] If the target of a ServiceReference has

2085 become unavailable, the SCA runtime SHOULD throw a ServiceUnavailableException when an

2086 operation is invoked on the ServiceReference. [JCA90032] If the target service of a

2087 ServiceReference is changed, the reference MAY continue to work, depending on the runtime and

2088 the type of change that was made. [JCA90033] If it doesn't work, the exception thrown will

2089 depend on the runtime and the cause of the failure.

2090 A reference or ServiceReference accessed through the component context by calling getService()

2091 or getServiceReference() MUST correspond to the current configuration of the domain. This applies

2092 whether or not reinjection has taken place. [JCA90034] If the target of a reference or

2093 ServiceReference accessed through the component context by calling getService() or

2094 getServiceReference() has been undeployed or has become unavailable, the result SHOULD be a

2095 reference to the undeployed or unavailable service, and attempts to call business methods

2096 SHOULD throw an InvalidServiceException or a ServiceUnavailableException. [JCA90035] If the

2097 target service of a reference or ServiceReference accessed through the component context by

2098 calling getService() or getServiceReference() has changed, the returned value SHOULD be a

2099 reference to the changed service. [JCA90036]

2100 The rules for reference reinjection also apply to references with a multiplicity of 0..n or 1..n. This

2101 means that in the cases where reference reinjection is not allowed, the array or Collection for a

2102 reference of multiplicity 0..n or multiplicity 1..n MUST NOT change its contents when changes

2103 occur to the reference wiring or to the targets of the wiring. [JCA90037] In cases where the

2104 contents of a reference array or collection change when the wiring changes or the targets change,

2105 then for references that use setter injection, the setter method MUST be called by the SCA

2106 runtime for any change to the contents. [JCA90038] A reinjected array or Collection for a

2107 reference MUST NOT be the same array or Collection object previously injected to the component.

2108 [JCA90039]

2109

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>			

2110

2111 9.18 @Remotable

2112 The following Java code defines the **@Remotable** annotation:

2113

```
2114 package org.oasisopen.sca.annotation;
2115
2116 import static java.lang.annotation.ElementType.TYPE;
2117 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2118 import java.lang.annotation.Retention;
2119 import java.lang.annotation.Target;
```

2120

```
2121
2122 @Target(TYPE)
2123 @Retention(RUNTIME)
2124 public @interface Remotable {
2125
2126 }
2127
```

2128 The @Remotable annotation is used to specify a Java service interface as remotable. A remotable
2129 service can be published externally as a service and MUST be translatable into a WSDL portType.
2130 [JCA90040]

2131 The @Remotable annotation has no attributes.

2132 The following snippet shows the Java interface for a remotable service with its @Remotable
2133 annotation.

Deleted: March

```

2134 package services.hello;
2135
2136 import org.oasisopen.sca.annotation.*;
2137
2138 @Remotable
2139 public interface HelloService {
2140     String hello(String message);
2141 }
2142
2143

```

2144 The style of remotable interfaces is typically **coarse grained** and intended for **loosely coupled**
2145 interactions. Remotable service interfaces are not allowed to make use of method **overloading**.

2146 Complex data types exchanged via remotable service interfaces need to be compatible with the
2147 marshalling technology used by the service binding. For example, if the service is going to be
2148 exposed using the standard Web Service binding, then the parameters can be JAXB [JAX-B] types
2149 or they can be Service Data Objects (SDOs) [SDO].

2150 Independent of whether the remotable service is called from outside of the composite that
2151 contains it or from another component in the same composite, the data exchange semantics are
2152 **by-value**.

2153 Implementations of remotable services can modify input data during or after an invocation and
2154 can modify return data after the invocation. If a remotable service is called locally or remotely, the
2155 SCA container is responsible for making sure that no modification of input data or post-invocation
2156 modifications to return data are seen by the caller.

2157 The following snippet shows a remotable Java service interface.

```

2158
2159 package services.hello;
2160
2161 import org.oasisopen.sca.annotation.*;
2162
2163 @Remotable
2164 public interface HelloService {
2165     String hello(String message);
2166 }
2167
2168 package services.hello;
2169
2170 import org.oasisopen.sca.annotation.*;
2171
2172 @Service(HelloService.class)
2173 public class HelloServiceImpl implements HelloService {
2174     public String hello(String message) {
2175         ...
2176     }
2177 }
2178
2179

```

2180 9.19 @Requires

2181 The following Java code defines the **@Requires** annotation:

```

2182 package org.oasisopen.sca.annotation;
2183
2184 import static java.lang.annotation.ElementType.FIELD;
2185

```

Deleted: March


```

2186 import static java.lang.annotation.ElementType.METHOD;
2187 import static java.lang.annotation.ElementType.PARAMETER;
2188 import static java.lang.annotation.ElementType.TYPE;
2189 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2190
2191 import java.lang.annotation.Inherited;
2192 import java.lang.annotation.Retention;
2193 import java.lang.annotation.Target;
2194
2195 @Inherited
2196 @Retention(RUNTIME)
2197 @Target({TYPE, METHOD, FIELD, PARAMETER})
2198 public @interface Requires {
2199     /**
2200      * Returns the attached intents.
2201      *
2202      * @return the attached intents
2203      */
2204     String[] value() default "";
2205 }
2206

```

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

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

2210 9.20 @Scope

2211 The following Java code defines the **@Scope** annotation:

```

2212 package org.oasisopen.sca.annotation;
2213
2214 import static java.lang.annotation.ElementType.TYPE;
2215 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2216 import java.lang.annotation.Retention;
2217 import java.lang.annotation.Target;
2218
2219 @Target(TYPE)
2220 @Retention(RUNTIME)
2221 public @interface Scope {
2222
2223     String value() default "STATELESS";
2224 }

```

2225 The @Scope annotation MUST only be used on a service's implementation class. It is an error to
2226 use this annotation on an interface. [JCA90041]

2227 The @Scope annotation has the following attribute:

- 2228 • **value** – the name of the scope.
2229 SCA defines the following scope names, but others can be defined by particular Java-
2230 based implementation types:
2231 STATELESS
2232 COMPOSITE
2233 For 'STATELESS' implementations, a different implementation instance can be used to
2234 service each request. Implementation instances can be newly created or be drawn from a
2235 pool of instances.

2236 The default value is STATELESS.

2237 The following snippet shows a sample for a COMPOSITE scoped service implementation:

```

2238 package services.hello;

```

Deleted: March

```

2239
2240 import org.oasisopen.sca.annotation.*;
2241
2242 @Service(HelloService.class)
2243 @Scope("COMPOSITE")
2244 public class HelloServiceImpl implements HelloService {
2245
2246     public String hello(String message) {
2247         ...
2248     }
2249 }
2250

```

9.21 @Service

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

```

2253 package org.oasisopen.sca.annotation;
2254
2255 import static java.lang.annotation.ElementType.TYPE;
2256 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2257 import java.lang.annotation.Retention;
2258 import java.lang.annotation.Target;
2259
2260 @Target(TYPE)
2261 @Retention(RUNTIME)
2262 public @interface Service {
2263
2264     Class<?>[] interfaces() default {};
2265     Class<?> value() default Void.class;
2266 }
2267

```

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]

2283

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

2288 **A component implementation MUST NOT have two services with the same Java simple name.**
2289 **[JCA90045]** If a Java implementation needs to realize two services with the same Java simple
2290 name then this can be achieved through subclassing of the interface.

2291 The following snippet shows an implementation of the HelloService marked with the @Service
2292 annotation.

```
2293 package services.hello;  
2294  
2295 import org.oasisopen.sca.annotation.Service;  
2296  
2297 @Service(HelloService.class)  
2298 public class HelloServiceImpl implements HelloService {  
2299     public void hello(String name) {  
2300         System.out.println("Hello " + name);  
2301     }  
2302 }  
2303  
2304
```

2305

10 WSDL to Java and Java to WSDL

2306
2307
2308

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.

2309
2310
2311
2312
2313
2314

~~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]~~

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.

2315
2316
2317
2318
2319
2320

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.

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.

2321
2322

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

- No support for holders

2323

2324
2325

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

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

2326

10.1 JAX-WS Client Asynchronous API for a Synchronous Service

2327
2328
2329
2330
2331

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.

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

2332
2333
2334
2335
2336
2337
2338
2339

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]

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

2340
2341

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

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

2342
2343
2344
2345

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

2346

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

2347

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

2348
2349
2350

- 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: March

2351 c. a return type of Future<?>
2352 then C is a JAX-WS callback method that isn't part of the SCA interface contract.
2353 As an example, an interface can be defined in WSDL as follows:

```
2354 <!-- WSDL extract -->  
2355 <message name="getPrice">  
2356   <part name="ticker" type="xsd:string"/>  
2357 </message>  
2358  
2359 <message name="getPriceResponse">  
2360   <part name="price" type="xsd:float"/>  
2361 </message>  
2362  
2363 <portType name="StockQuote">  
2364   <operation name="getPrice">  
2365     <input message="tns:getPrice"/>  
2366     <output message="tns:getPriceResponse"/>  
2367   </operation>  
2368 </portType>
```

2369
2370 The JAX-WS asynchronous mapping will produce the following Java interface:

```
2371 // asynchronous mapping  
2372 @WebService  
2373 public interface StockQuote {  
2374   float getPrice(String ticker);  
2375   Response<Float> getPriceAsync(String ticker);  
2376   Future<?> getPriceAsync(String ticker, AsyncHandler<Float>);  
2377 }
```

2378
2379 For SCA interface definition purposes, this is treated as equivalent to the following:

```
2380 // synchronous mapping  
2381 @WebService  
2382 public interface StockQuote {  
2383   float getPrice(String ticker);  
2384 }
```

2385
2386 **SCA runtimes MUST support the use of the JAX-WS client asynchronous model. [JCA100009]** In
2387 the above example, if the client implementation uses the asynchronous form of the interface, the
2388 two additional getPriceAsync() methods can be used for polling and callbacks as defined by the
2389 JAX-WS specification.

2390

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

```
2391 <?xml version="1.0" encoding="UTF-8"?>
2392 <!-- (c) Copyright SCA Collaboration 2006 -->
2393 <schema xmlns="http://www.w3.org/2001/XMLSchema"
2394         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2395         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2396         elementFormDefault="qualified">
2397
2398     <include schemaLocation="sca-core.xsd"/>
2399
2400     <element name="interface.java" type="sca:JavaInterface"
2401             substitutionGroup="sca:interface"/>
2402     <complexType name="JavaInterface">
2403         <complexContent>
2404             <extension base="sca:Interface">
2405                 <sequence>
2406                     <any namespace="##other" processContents="lax"
2407                         minOccurs="0" maxOccurs="unbounded"/>
2408                 </sequence>
2409                 <attribute name="interface" type="NCName" use="required"/>
2410                 <attribute name="callbackInterface" type="NCName"
2411                             use="optional"/>
2412                 <anyAttribute namespace="##any" processContents="lax"/>
2413             </extension>
2414         </complexContent>
2415     </complexType>
2416 </schema>
2417
```

2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
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

B. Java Classes and Interfaces

B.1 SCAClient Classes and Interfaces

B.1.1 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.NoSuchDomainException;  
import org.oasisopen.sca.NoSuchServiceException;  
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;
```

Deleted: D

Formatted: AppendixHeading2

Formatted: Bullets and Numbering

Deleted: March

```

2468 /**
2469 * Prevent concrete subclasses from using the no-arg constructor
2470 */
2471 private SCAClientFactory() {
2472 }
2473
2474 /**
2475 * Constructor used by concrete subclasses
2476 * @param domainURI - The Domain URI of the Domain accessed via this
2477 * SCAClientFactory
2478 */
2479 protected SCAClientFactory(URI domainURI) {
2480     this.domainURI = domainURI;
2481 }
2482
2483 /**
2484 * Gets the Domain URI of the Domain accessed via this SCAClientFactory
2485 * @return - the URI for the Domain
2486 */
2487 protected URI getDomainURI() {
2488     return domainURI;
2489 }
2490
2491
2492 /**
2493 * Creates a new instance of the SCAClient that can be
2494 * used to lookup SCA Services.
2495 *
2496 * @param domainURI      URI of the target domain for the SCAClient
2497 * @return A new SCAClient
2498 */
2499 public static SCAClientFactory newInstance( URI domainURI )
2500     throws NoSuchDomainException {
2501     return newInstance(null, null, domainURI);
2502 }
2503
2504 /**
2505 * Creates a new instance of the SCAClient that can be
2506 * used to lookup SCA Services.
2507 *
2508 * @param properties    Properties that may be used when
2509 * creating a new instance of the SCAClient
2510 * @param domainURI     URI of the target domain for the SCAClient
2511 * @return A new SCAClient instance
2512 */
2513 public static SCAClientFactory newInstance(Properties properties,
2514                                            URI domainURI)
2515     throws NoSuchDomainException {
2516     return newInstance(properties, null, domainURI);
2517 }
2518
2519 /**
2520 * Creates a new instance of the SCAClient that can be
2521 * used to lookup SCA Services.
2522 *
2523 * @param classLoader   ClassLoader that may be used when
2524 * creating a new instance of the SCAClient
2525 * @param domainURI     URI of the target domain for the SCAClient

```

Deleted: March


```

2526     * @return A new SCAClient instance
2527     */
2528     public static SCAClientFactory newInstance(ClassLoader classLoader,
2529                                               URI domainURI)
2530     throws NoSuchDomainException {
2531         return newInstance(null, classLoader, domainURI);
2532     }
2533
2534     /**
2535     * Creates a new instance of the SCAClient that can be
2536     * used to lookup SCA Services.
2537     *
2538     * @param properties Properties that may be used when
2539     * creating a new instance of the SCAClient
2540     * @param classLoader ClassLoader that may be used when
2541     * creating a new instance of the SCAClient
2542     * @param domainURI URI of the target domain for the SCAClient
2543     * @return A new SCAClient instance
2544     */
2545     public static SCAClientFactory newInstance(Properties properties,
2546                                               ClassLoader classLoader,
2547                                               URI domainURI)
2548     throws NoSuchDomainException {
2549         final SCAClientFactoryFinder finder =
2550             factoryFinder != null ? factoryFinder :
2551             new SCAClientFactoryFinderImpl();
2552         final SCAClientFactory factory
2553             = finder.find(properties, classLoader, domainURI);
2554         return factory;
2555     }
2556
2557     /**
2558     * Returns a reference proxy that implements the business interface <T>
2559     * of a service in the SCA Domain handled by this SCAClientFactory
2560     *
2561     * @param serviceURI the relative URI of the target service. Takes the
2562     * form componentName/serviceName.
2563     * Can also take the extended form componentName/serviceName/bindingName
2564     * to use a specific binding of the target service
2565     *
2566     * @param interfaze The business interface class of the service in the
2567     * domain
2568     * @param <T> The business interface class of the service in the domain
2569     *
2570     * @return a proxy to the target service, in the specified SCA Domain
2571     * that implements the business interface <B>.
2572     * @throws NoSuchServiceException Service requested was not found
2573     * @throws NoSuchDomainException Domain requested was not found
2574     */
2575     public abstract <T> T getService(Class<T> interfaze, String serviceURI)
2576     throws NoSuchServiceException, NoSuchDomainException;
2577 }

```

2578 B.1.2 SCAClientFactoryFinder interface

2579 The SCAClientFactoryFinder interface is a Service Provider Interface representing a SCAClientFactory
2580 finder. SCA provides a default reference implementation of this interface. SCA runtime vendors can
2581 create alternative implementations of this interface that use different class loading or lookup mechanisms.

Deleted: protected

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

Formatted: Bullets and Numbering

Formatted: Normal

Deleted: March

```

2582
2583 /*
2584  * Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
2585  * OASIS trademark, IPR and other policies apply.
2586  */
2587
2588 package org.oasisopen.sca.client;
2589
2590 import java.net.URI;
2591 import java.util.Properties;
2592
2593 import org.oasisopen.sca.NoSuchDomainException;
2594
2595 /* A Service Provider Interface representing a SCAClientFactory finder.
2596  * SCA provides a default reference implementation of this interface.
2597  * SCA runtime vendors can create alternative implementations of this
2598  * interface that use different class loading or lookup mechanisms.
2599  */
2600 public interface SCAClientFactoryFinder {
2601
2602     /**
2603      * Method for finding the SCAClientFactory for a given Domain URI using
2604      * a specified set of properties and a a specified ClassLoader
2605      * @param properties - properties to use - may be null
2606      * @param classLoader - ClassLoader to use - may be null
2607      * @param domainURI - the Domain URI - must be a valid SCA Domain URI
2608      * @return - the SCAClientFactory or null if the factory could not be
2609      * @throws - NoSuchDomainException if the domainURI does not reference
2610      * a valid SCA Domain
2611      * found
2612      */
2613     SCAClientFactory find(Properties properties,
2614                           ClassLoader classLoader,
2615                           URI domainURI )
2616     throws NoSuchDomainException ;
2617 }

```

← Formatted: Normal

2618 **B.1.3 SCAClientFactoryFinderImpl class**

2619 [This class provides a default implementation for finding a provider's SCAClientFactory implementation](#)
2620 [class. It is used if the provider does not inject its SCAClientFactoryFinder implementation class into the](#)
2621 [base SCAClientFactory class.](#)

2622 [It discovers a provider's SCAClientFactory implementation by referring to the following information in this](#)
2623 [order:](#)

- 2624 1. [The org.oasisopen.sca.client.SCAClientFactory property from the Properties specified on the](#)
2625 [newInstance\(\) method call if specified](#)
- 2626 2. [The org.oasisopen.sca.client.SCAClientFactory property from the System Properties](#)
- 2627 3. [The META-INF/services/org.oasisopen.sca.client.SCAClientFactory file](#)

```

2628 /*
2629  * Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
2630  * OASIS trademark, IPR and other policies apply.
2631  */
2632 package org.oasisopen.sca.client.impl;
2633
2634 import org.oasisopen.sca.client.SCAClientFactoryFinder;
2635

```

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.¶

Deleted: March

```

2636 import java.io.BufferedReader;
2637 import java.io.Closeable;
2638 import java.io.IOException;
2639 import java.io.InputStream;
2640 import java.io.InputStreamReader;
2641 import java.lang.reflect.Constructor;
2642 import java.net.URI;
2643 import java.net.URL;
2644 import java.util.Properties;
2645
2646 import org.oasisopen.sca.NoSuchDomainException;
2647 import org.oasisopen.sca.ServiceRuntimeException;
2648 import org.oasisopen.sca.client.SCAClientFactory;
2649
2650 /**
2651  * This is a default implementation of an SCAClientFactoryFinder which is
2652  * used to find an implementation of the SCAClientFactory interface.
2653  *
2654  * @see SCAClientFactoryFinder
2655  * @see SCAClientFactory
2656  *
2657  * @author OASIS Open
2658  */
2659 public class SCAClientFactoryFinderImpl implements SCAClientFactoryFinder {
2660
2661     /**
2662      * The name of the System Property used to determine the SPI
2663      * implementation to use for the SCAClientFactory.
2664      */
2665     private static final String SCA_CLIENT_FACTORY_PROVIDER_KEY =
2666         SCAClientFactory.class.getName();
2667
2668     /**
2669      * The name of the file loaded from the ClassPath to determine
2670      * the SPI implementation to use for the SCAClientFactory.
2671      */
2672     private static final String SCA_CLIENT_FACTORY_PROVIDER_META_INF_SERVICE
2673         = "META-INF/services/" + SCA_CLIENT_FACTORY_PROVIDER_KEY;
2674
2675     /**
2676      * Public Constructor
2677      */
2678     public SCAClientFactoryFinderImpl() {
2679     }
2680
2681     /**
2682      * Creates an instance of the SCAClientFactorySPI implementation.
2683      * This discovers the SCAClientFactorySPI Implementation and instantiates
2684      * the provider's implementation.
2685      *
2686      * @param properties Properties that may be used when creating a new
2687      * instance of the SCAClient
2688      * @param classLoader ClassLoader that may be used when creating a new
2689      * instance of the SCAClient
2690      * @return new instance of the SCAClientFactory
2691      * @throws ServiceRuntimeException Failed to create SCAClientFactory
2692      * Implementation.
2693      */

```

```

2694     public SCAClientFactory find(Properties properties,
2695                               ClassLoader classLoader,
2696                               URI domainURI )
2697     throws NoSuchDomainException, ServiceRuntimeException {
2698         if (classLoader == null) {
2699             classLoader = getThreadContextClassLoader ();
2700         }
2701         final String factoryImplClassName =
2702             discoverProviderFactoryImplClass(properties, classLoader);
2703         final Class<? extends SCAClientFactory> factoryImplClass
2704             = loadProviderFactoryClass(factoryImplClassName,
2705                                       classLoader);
2706         final SCAClientFactory factory =
2707             instantiateSCAClientFactoryClass(factoryImplClass,
2708                                             domainURI );
2709         return factory;
2710     }
2711
2712     /**
2713     * Gets the Context ClassLoader for the current Thread.
2714     *
2715     * @return The Context ClassLoader for the current Thread.
2716     */
2717     private static ClassLoader getThreadContextClassLoader () {
2718         final ClassLoader threadClassLoader =
2719             Thread.currentThread().getContextClassLoader();
2720         return threadClassLoader;
2721     }
2722
2723     /**
2724     * Attempts to discover the class name for the SCAClientFactorySPI
2725     * implementation from the specified Properties, the System Properties
2726     * or the specified ClassLoader.
2727     *
2728     * @return The class name of the SCAClientFactorySPI implementation
2729     * @throw ServiceRuntimeException Failed to find implementation for
2730     * SCAClientFactorySPI.
2731     */
2732     private static String
2733         discoverProviderFactoryImplClass(Properties properties,
2734                                       ClassLoader classLoader)
2735     throws ServiceRuntimeException {
2736         String providerClassName =
2737             checkPropertiesForSPIClassName(properties);
2738         if (providerClassName != null) {
2739             return providerClassName;
2740         }
2741
2742         providerClassName =
2743             checkPropertiesForSPIClassName(System.getProperties());
2744         if (providerClassName != null) {
2745             return providerClassName;
2746         }
2747
2748         providerClassName = checkMETAINFOServicesForSIPClassName(classLoader);
2749         if (providerClassName == null) {
2750             throw new ServiceRuntimeException(
2751                 "Failed to find implementation for SCAClientFactory");
2752         }

```

Deleted: March

```

2752     }
2753
2754     return providerClassName;
2755 }
2756
2757 /**
2758  * Attempts to find the class name for the SCAClientFactorySPI
2759  * implementation from the specified Properties.
2760  *
2761  * @return The class name for the SCAClientFactorySPI implementation
2762  * or <code>null</code> if not found.
2763  */
2764 private static String
2765 checkPropertiesForSPIClassName(Properties properties) {
2766     if (properties == null) {
2767         return null;
2768     }
2769
2770     final String providerClassName =
2771         properties.getProperty(SCA_CLIENT_FACTORY_PROVIDER_KEY);
2772     if (providerClassName != null && providerClassName.length() > 0) {
2773         return providerClassName;
2774     }
2775
2776     return null;
2777 }
2778
2779 /**
2780  * Attempts to find the class name for the SCAClientFactorySPI
2781  * implementation from the META-INF/services directory
2782  *
2783  * @return The class name for the SCAClientFactorySPI implementation or
2784  * <code>null</code> if not found.
2785  */
2786 private static String checkMETAINFServicesForSIPClassName(ClassLoader cl)
2787 {
2788     final URL url =
2789         cl.getResource(SCA_CLIENT_FACTORY_PROVIDER_META_INF_SERVICE);
2790     if (url == null) {
2791         return null;
2792     }
2793
2794     InputStream in = null;
2795     try {
2796         in = url.openStream();
2797         BufferedReader reader = null;
2798         try {
2799             reader =
2800                 new BufferedReader(new InputStreamReader(in, "UTF-8"));
2801
2802             String line;
2803             while ((line = readNextLine(reader)) != null) {
2804                 if (!line.startsWith("#") && line.length() > 0) {
2805                     return line;
2806                 }
2807             }
2808
2809             return null;

```

Deleted: March

```

2810         } finally {
2811             closeStream(reader);
2812         }
2813     } catch (IOException ex) {
2814         throw new ServiceRuntimeException(
2815             "Failed to discover SCAClientFactory provider", ex);
2816     } finally {
2817         closeStream(in);
2818     }
2819 }
2820
2821 /**
2822  * Reads the next line from the reader and returns the trimmed version
2823  * of that line
2824  *
2825  * @param reader The reader from which to read the next line
2826  * @return The trimmed next line or <code>null</code> if the end of the
2827  * stream has been reached
2828  * @throws IOException I/O error occurred while reading from Reader
2829  */
2830 private static String readNextLine(BufferedReader reader)
2831     throws IOException {
2832
2833     String line = reader.readLine();
2834     if (line != null) {
2835         line = line.trim();
2836     }
2837     return line;
2838 }
2839
2840 /**
2841  * Loads the specified SCAClientFactory Implementation class.
2842  *
2843  * @param factoryImplClassName The name of the SCAClientFactory
2844  * Implementation class to load
2845  * @return The specified SCAClientFactory Implementation class
2846  * @throws ServiceRuntimeException Failed to load the SCAClientFactory
2847  * Implementation class
2848  */
2849 private static Class<? extends SCAClientFactory>
2850     loadProviderFactoryClass(String factoryImplClassName,
2851                             ClassLoader classLoader)
2852     throws ServiceRuntimeException {
2853
2854     try {
2855         final Class<?> providerClass =
2856             classLoader.loadClass(factoryImplClassName);
2857         final Class<? extends SCAClientFactory> providerFactoryClass =
2858             providerClass.asSubclass(SCAClientFactory.class);
2859         return providerFactoryClass;
2860     } catch (ClassNotFoundException ex) {
2861         throw new ServiceRuntimeException(
2862             "Failed to load SCAClientFactory implementation class "
2863             + factoryImplClassName, ex);
2864     } catch (ClassCastException ex) {
2865         throw new ServiceRuntimeException(
2866             "Loaded SCAClientFactory implementation class "
2867             + factoryImplClassName

```

Deleted: March

```

2868         + " is not a subclass of "
2869         + SCAClientFactory.class.getName() , ex);
2870     }
2871 }
2872
2873 /**
2874  * Instantiate an instance of the specified SCAClientFactorySPI
2875  * Implementation class.
2876  *
2877  * @param factoryImplClass The SCAClientFactorySPI Implementation
2878  * class to instantiate.
2879  * @return An instance of the SCAClientFactorySPI Implementation class
2880  * @throws ServiceRuntimeException Failed to instantiate the specified
2881  * specified SCAClientFactorySPI Implementation class
2882  */
2883 private static SCAClientFactory instantiateSCAClientFactoryClass(
2884     Class<? extends SCAClientFactory> factoryImplClass,
2885     URI domainURI)
2886     throws NoSuchDomainException, ServiceRuntimeException {
2887
2888     try {
2889         Constructor<? extends SCAClientFactory> URIConstructor =
2890             factoryImplClass.getConstructor(domainURI.getClass());
2891         SCAClientFactory provider =
2892             URIConstructor.newInstance( domainURI );
2893         return provider;
2894     } catch (Throwable ex) {
2895         throw new ServiceRuntimeException(
2896             "Failed to instantiate SCAClientFactory implementation class "
2897             + factoryImplClass, ex);
2898     }
2899 }
2900
2901 /**
2902  * Utility method for closing Closeable Object.
2903  *
2904  * @param closeable The Object to close.
2905  */
2906 private static void closeStream(Closeable closeable) {
2907     if (closeable != null) {
2908         try{
2909             closeable.close();
2910         } catch (IOException ex) {
2911             throw new ServiceRuntimeException("Failed to close stream",
2912                 ex);
2913         }
2914     }
2915 }
2916 }

```

2917 **B.1.4 SCAClient Classes and Interfaces - what does a vendor need to do?**

2918 [The SCAClient classes and interfaces are designed so that vendors can provide their own](#)
2919 [implementation suited to the needs of their SCA runtime. This section describes the tasks that a vendor](#)
2920 [needs to consider in relation to the SCAClient classes and interfaces.](#)

- 2921 • [Implement their SCAClientFactory implementation class](#)

2922

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

Deleted: March

2923 [Vendors need to provide an subclass of SCAClientFactory that is capable of looking up Services](#)
2924 [in their SCA Runtime. Vendors need to subclass SCAClientFactory and implement the](#)
2925 [getService\(\) method so that it creates an reference proxies to services in SCA Domains handled](#)
2926 [by their SCA runtime\(s\).](#)

2927

2928

2929 • [Configure the Vendor SCAClientFactory implementation class so that it gets used](#)
2930 [Vendors have several options:](#)

2931

2932 [Option 1: Set System Property to point to the Vendor's implementation](#)

2933

2934 [Vendors set the org.oasisopen.sca.client.SCAClientFactory System Property to point to their](#)
2935 [implementation class and use the reference implementation of SCAClientFactoryFinder](#)

2936

2937 [Option 2: Provide a META-INF/services file](#)

2938

2939 [Vendors provide a META-INF/services/org.oasisopen.sca.client.SCAClientFactory file that points](#)
2940 [to their implementation class and use the reference implementation of SCAClientFactoryFinder](#)

2941

2942 [Option 3: Inject a vendor implementation of the SCAClientFactoryFinder interface into](#)
2943 [SCAClientFactory](#)

2944

2945 [Vendors inject an instance of the vendor implementation of SCAClientFactoryFinder into the](#)
2946 [factoryFinder field of the SCAClientFactory abstract class. The reference implementation of](#)
2947 [SCAClientFactoryFinder is not used in this scenario. The vendor implementation of](#)
2948 [SCAClientFactoryFinder can find the vendor implementation\(s\) of SCAClientFactory by any](#)
2949 [means.](#)

2950

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.

2951

C. Conformance Items

2952 This section contains a list of conformance items for the SCA Java Common Annotations and APIs
2953 specification.

2954

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.

[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.

2955

Deleted: March

2956 **D. Acknowledgements**

2957 The following individuals have participated in the creation of this specification and are gratefully
2958 acknowledged:

2959 **Participants:**

2960 [Participant Name, Affiliation | Individual Member]

2961 [Participant Name, Affiliation | Individual Member]

2962

Deleted: March

E. Non-Normative Text

Deleted: March

2964

F. Revision History

2965 [optional; should not be included in OASIS Standards]

2966

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 Combellack	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

2967

Deleted: March

Page 53: [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 53: [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 53: [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 53: [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 53: [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 54: [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 54: [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 54: [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 54: [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 54: [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 54: [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 54: [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 54: [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 54: [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 54: [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 54: [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 54: [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.