Service Component Architecture SCA-J Common Annotations and APIs Specification Version 1.1

Committee Draft 03 – Rev2

19 January 2010

Deleted: 4
Deleted: 6
Deleted: December
Deleted: 09

Specification URIs:

This Version:

http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd03.html
http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd03.doc
http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd03.pdf (Authoritative)

Previous Version:

http://www.oasis-open.org/apps/org/workgroup/sca-j/download.php/30880/sca-javacaa-1.1-spec-cd02.doc
http://www.oasis-open.org/apps/org/workgroup/sca-j/download.php/31427/sca-javacaa-1.1-spec-cd02.pdf (Authoritative)

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 (Authoritative)

Latest Approved Version:

Technical Committee:

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

Chair(s):

David Booz, IBM
Mark Combellack, Avaya

Editor(s):

David Booz, IBM
Mark Combellack, Avaya
Mike Edwards, IBM
Anish Karmarkar, Oracle

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

Declared XML Namespace(s):

http://docs.oasis-open.org/ns/opencsa/sca/200912

Deleted: 200903
Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

Abstract:

The SCA-J Common Annotations and APIs specification defines 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 artifacts described by other SCA specifications such as the POJO Component Implementation Specification [JAVA_CI].

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 other Java-based SCA specifications can choose 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/.

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

Notices

Copyright © OASIS® 2005, 2010. All Rights Reserved.

Deleted: 09

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

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

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

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

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

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

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

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

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

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
2.3 @AllowsPassByReference ... 11
2.3.1 Marking Services and References as “allows pass by reference” ... 12
2.3.2 Applying “allows pass by reference” to Service Proxies ... 12
2.3.3 Using “allows pass by reference” to Optimize Remotable Calls ... 13
3 Interface ... 14
3.1 Java Interface Element – <interface.java> ... 14
3.2 @Remotable ... 15
3.3 @Callback ... 15
3.4 @AsyncInvocation ... 15
3.5 SCA Java Annotations for Interface Classes ... 16
3.6 Compatibility of Java Interfaces ... 16
4 SCA Component Implementation Lifecycle ... 17
4.1 Overview of SCA Component Implementation Lifecycle ... 17
4.2 SCA Component Implementation Lifecycle State Diagram ... 17
4.2.1 Constructing State ... 18
4.2.2 Injecting State ... 18
4.2.3 Initializing State ... 19
4.2.4 Running State ... 19
4.2.5 Destroying State ... 19
4.2.6 Terminated State ... 20
5 Client API ... 21
5.1 Accessing Services from an SCA Component ... 21
5.1.1 Using the Component Context API ... 21
5.2 Accessing Services from non-SCA Component Implementations ... 21
5.2.1 SCAClientFactory Interface and Related Classes ... 21
6 Error Handling ... 23
7 Asynchronous Programming ... 24
7.1 @OneWay ... 24
7.2 Callbacks ... 24

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

7.2.1 Using Callbacks 24
7.2.2 Callback Instance Management 26
7.2.3 Callback Injection 26
7.2.4 Implementing Multiple Bidirectional Interfaces 26
7.2.5 Accessing Callbacks 27
7.3 Asynchronous handling of Long Running Service Operations 28
7.3.1 SCA Asynchronous Service Interface 28
8 Policy Annotations for Java 31
8.1 General Intent Annotations 31
8.2 Specific Intent Annotations 33
8.2.1 How to Create Specific Intent Annotations 33
8.3 Application of Intent Annotations 34
8.3.1 Intent Annotation Examples 34
8.3.2 Inheritance and Annotation 36
8.4 Relationship of Declarative and Annotated Intents 37
8.5 Policy Set Annotations 37
8.6 Security Policy Annotations 39
8.7 Transaction Policy Annotations 40
9 Java API 41
9.1 Component Context 41
9.2 Request Context 43
9.3 ServiceReference 43
9.4 ServiceRuntimeException 44
9.5 ServiceUnavailableException 45
9.6 InvalidServiceException 45
9.7 Constants 46
9.8 SCAClientFactory Class 46
9.9 SCAClientFactoryFinder Interface 49
9.10 SCAClientFactoryFinderImpl Class 49
9.11 NoSuchDomainException 50
9.12 NoSuchServiceException 50
10 Java Annotations 52
10.1 @AllowsPassByReference 52
10.2 @AsyncInvocation 53
10.3 @Authentication 54
10.4 @Authorization 55
10.5 @Callback 55
10.6 @ComponentName 57
10.7 @Confidentiality 57
10.8 @Constructor 58
10.9 @Context 59
10.10 @Destroy 59
10.11 @EagerInit 60
10.12 @Init 60
10.13 @Integrity 61

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

10.14 @Intent 61
10.15 @ManagedSharedTransaction........ 62
10.16 @ManagedTransaction 63
10.17 @MutualAuthentication 63
10.18 @NoManagedTransaction 64
10.19 @OneWay 65
10.20 @PolicySets........ 65
10.21 @Property........ 66
10.22 @Qualifier 67
10.23 @Reference........ 68
10.23.1 Reinjection 70
10.24 @Remotable 72
10.25 @Requires 74
10.26 @Scope 74
10.27 @Service 75
11 WSDL to Java and Java to WSDL 77
11.1 JAX-WS Client Asynchronous API for a Synchronous Service........ 80
12 Conformance 83
12.1 SCA Java XML Document........ 83
12.2 SCA Java Class 83
12.3 SCA Runtime 83
A. XML Schema: sca-interface-java.xsd........ 84
B. Java Classes and Interfaces........ 85
B.1 SCAClient Classes and Interfaces........ 85
B.1.1 SCAClientFactory Class 85
B.1.2 SCAClientFactoryFinder interface 87
B.1.3 SCAClientFactoryFinderImpl class 88
B.1.4 SCAClient Classes and Interfaces - what does a vendor need to do? 93
C. Conformance Items........ 95
D. Acknowledgements........ 105
E. Non-Normative Text........ 107
F. Revision History........ 108

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

1 Introduction

The SCA-J Common Annotations and APIs specification defines 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 SCA Java-based 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

The goal of defining the annotations and APIs in this specification is to promote consistency and reduce duplication across the various SCA Java-based specifications. The annotations and APIs defined in this specification are designed to be used by other SCA Java-based specifications in either a partial or complete fashion.

1.1 Terminology

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

1.2 Normative References

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

[ASSEMBLY] SCA Assembly Model Specification Version 1.1, http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.pdf

[JAVA_CI] SCA POJO Component Implementation Specification Version 1.1 http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-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,

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

[JSR-250] Common Annotations for the Java Platform specification (JSR-250), http://www.jcp.org/en/jsr/detail?id=250

[JAX-WS] JAX-WS 2.1 Specification (JSR-224), http://www.jcp.org/en/jsr/detail?id=224

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

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

44 **[JAAS]** Java Authentication and Authorization Service Reference Guide
45 http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.
46 html

47 1.3 Non-Normative References

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

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

2 Implementation Metadata

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

2.1 Service Metadata

2.1.1 @Service

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

- As a Java interface
- As a Java class
- As a Java interface generated from a Web Services Description Language [WSDL] (WSDL) portType (Java interfaces generated from WSDL portTypes are always ***remotable***)

2.1.2 Java Semantics of a Remotable Service

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

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

```
package services.hello;
@Remotable
public interface HelloService {
   String hello(String message);
}
```

2.1.3 Java Semantics of a Local Service

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

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

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

```
package services.hello;
public interface HelloService {
   String hello(String message);
}
```

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

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

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

89 ### 2.1.4 @Reference

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

93 ### 2.1.5 @Property

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

97 ## 2.2 Implementation Scopes: @Scope, @Init, @Destroy

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

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

104 This specification defines two scopes:

105 - STATELESS
106 - COMPOSITE

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

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

111 ***@Init*** denotes a method called upon first use of an instance during the lifetime of the scope
112 (except for composite scoped implementation marked to eagerly initialize, see section Composite
113 Scope).

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

115 Note that only no-argument methods with a void return type can be annotated as lifecycle
116 methods.

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

```
119
120    @Init
121    public void start() {
122          ...
123    }
124
125    @Destroy
126    public void stop() {
127          ...
128    }
129
```

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

132 ### 2.2.1 Stateless Scope

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

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

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

2.2.2 Composite Scope

The meaning of "composite scope" is defined in relation to the composite containing the component.

It is important to distinguish between different uses of a composite, where these uses affect the numbers of instances of components within the composite. There are 2 cases:

a) Where the composite containing the component using the Java implementation is the SCA Domain (i.e. a deployment composite declares the component using the implementation)

b) Where the composite containing the component using the Java implementation is itself used as the implementation of a higher level component (any level of nesting is possible, but the component is NOT at the Domain level)

Where an implementation is used by a "domain level component", and the implementation is marked "Composite" scope, the SCA runtime MUST ensure that all consumers of the component appear to be interacting with a single runtime instance of the implementation. [JCA20004]

Where an implementation is marked "Composite" scope and it is used by a component that is nested inside a composite that is used as the implementation of a higher level component, the SCA runtime MUST ensure that all consumers of the component appear to be interacting with a single runtime instance of the implementation. There can be multiple instances of the higher level component, each running on different nodes in a distributed SCA runtime. [JCA20008]

The SCA runtime can exploit shared state technology in combination with other well known high availability techniques to provide the appearance of a single runtime instance for consumers of composite scoped components.

The lifetime of the containing composite is defined as the time it becomes active in the runtime to the time it is deactivated, either normally or abnormally.

When the implementation class is marked for eager initialization, the SCA runtime MUST create a composite scoped instance when its containing component is started. [JCA20005] 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. [JCA20006]

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

2.3 @AllowsPassByReference

Calls to remotable services (see section "Java Semantics of a Remotable Service") have by-value semantics. This means that input parameters passed to the service can be modified by the service without these modifications being visible to the client. Similarly, the return value or exception from the service can be modified by the client without these modifications being visible to the service implementation. For remote calls (either cross-machine or cross-process), these semantics are a consequence of marshalling input parameters, return values and exceptions "on the wire" and unmarshalling them "off the wire" which results in physical copies being made. For local method calls within the same JVM, Java language calling semantics are by-reference and therefore do not provide the correct by-value semantics for SCA remotable interfaces. To compensate for this, the SCA runtime can intervene in these calls to provide by-value semantics by making copies of any mutable objects passed.

The cost of such copying can be very high relative to the cost of making a local call, especially if the data being passed is large. Also, in many cases this copying is not needed if the

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

185 implementation observes certain conventions for how input parameters, return values and
186 exceptions are used. The @AllowsPassByReference annotation allows service method
187 implementations and client references to be marked as “allows pass by reference” to indicate that
188 they use input parameters, return values and exceptions in a manner that allows the SCA runtime
189 to avoid the cost of copying mutable objects when a remotable service is called locally within the
190 same JVM.

191 2.3.1 Marking Services and References as “allows pass by reference”

192 Marking a service method implementation as “allows pass by reference” asserts that the method
193 implementation observes the following restrictions:

194 • Method execution will not modify any input parameter before the method returns.

195 • The service implementation will not retain a reference to any mutable input parameter,
196 mutable return value or mutable exception after the method returns.

197 • The method will observe “allows pass by value” client semantics (see below) for any
198 callbacks that it makes.

199 See section "@AllowsPassByReference" for details of how the @AllowsPassByReference annotation
200 is used to mark a service method implementation as “allows pass by reference”.

201 Marking a client reference as “allows pass by reference” asserts that method calls through the
202 reference observe the following restrictions:

203 • The client implementation will not modify any of the method’s input parameters before
204 the method returns. Such modifications might occur in callbacks or separate client
205 threads.

206 • If the method is one-way, the client implementation will not modify any of the method’s
207 input parameters at any time after calling the method. This is because one-way method
208 calls return immediately without waiting for the service method to complete.

209 See section "Applying “allows pass by reference” to Service Proxies" for details of how the
210 @AllowsPassByReference annotation is used to mark a client reference as “allows pass by
211 reference”.

212 2.3.2 Applying “allows pass by reference” to Service Proxies

213 Service method calls are made by clients using service proxies, which can be obtained by injection
214 into client references or by making API calls. A service proxy is marked as “allows pass by
215 reference” if and only if any of the following applies:

216 • It is injected into a reference or callback reference that is marked “allows pass by
217 reference”.

218 • It is obtained by calling ComponentContext.getService() or
219 ComponentContext.getServices() with the name of a reference that is marked “allows
220 pass by reference”.

221 • It is obtained by calling RequestContext.getCallback() from a service implementation that
222 is marked “allows pass by reference”.

223 • It is obtained by calling ServiceReference.getService() on a service reference that is
224 marked “allows pass by reference” (see definition below).

225 A service reference for a remotable service call is marked “allows pass by reference” if and only if
226 any of the following applies:

227 • It is injected into a reference or callback reference that is marked “allows pass by
228 reference”.

229 • It is obtained by calling ComponentContext.getServiceReference() or
230 ComponentContext.getServiceReferences() with the name of a reference that is marked
231 “allows pass by reference”.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

232
233
- It is obtained by calling RequestContext.getCallbackReference() from a service implementation that is marked "allows pass by reference".

234
235
- It is obtained by calling ComponentContext.cast() on a proxy that is marked "allows pass by reference".

236
2.3.3 Using "allows pass by reference" to Optimize Remotable Calls

237
238
239
240
The SCA runtime MAY use by-reference semantics when passing input parameters, return values or exceptions on calls to remotable services within the same JVM if both the service method implementation and the service proxy used by the client are marked "allows pass by reference". [JCA20009]

241
242
243
244
The SCA runtime MUST use by-value semantics when passing input parameters, return values and exceptions on calls to remotable services within the same JVM if the service method implementation is not marked "allows pass by reference" or the service proxy used by the client is not marked "allows pass by reference". [JCA20010]

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

3 Interface

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

3.1 Java Interface Element – <interface.java>

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

It is possible that the Java interface class referenced by the <interface.java/> element contains one or more annotations defined by the JAX-WS specification [JAX-WS]. These annotations can affect the interpretation of the <interface.java/> element. In the most extreme case, the annotations cause the replacement of the <interface.java/> element with an <interface.wsdl/> element. The relevant JAX-WS annotations and their effects on the <interface.java/> element are described in the section "JAX-WS Annotations and SCA Interfaces".

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

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

```
<interface.java interface="NCName" callbackInterface="NCName"?
                requires="list of xs:QName"?
                policySets="list of xs:QName"?
                remotable="boolean"?/>
```

The interface.java element has the following attributes:

- ***interface : NCName (1..1)*** – the Java interface class to use for the service interface. The value of the @interface attribute MUST be the fully qualified name of the Java interface class [JCA30001]

  If the identified class is annotated with either the JAX-WS @WebService or @WebServiceProvider annotations and the annotation has a non-empty ***wsdlLocation*** property, then the SCA Runtime MUST act as if an <interface.wsdl/> element is present instead of the <interface.java/> element, with an @interface attribute identifying the portType mapped from the Java interface class and containing @requires and @policySets attribute values equal to the @requires and @policySets attribute values of the <interface.java/> element. [JCA30010]
- ***callbackInterface : NCName (0..1)*** – the Java interface class to use for the callback interface. The value of the @callbackInterface attribute MUST be the fully qualified name of a Java interface used for callbacks [JCA30002]
- ***requires : QName (0..1)*** – a list of policy intents. See the Policy Framework specification [POLICY] for a description of this attribute
- ***policySets : QName (0..1)*** – a list of policy sets. See the Policy Framework specification [POLICY] for a description of this attribute.
- ***remotable : boolean (0..1)*** – indicates whether or not the interface is remotable. A value of "true" means the interface is remotable and a value of "false" means it is not. This attribute does not have a default value. If it is not specified then the remotability is determined by the presence or absence of the @Remotable annotation on the interface class. The @remotable attribute applies to both the interface and any optional callbackInterface. The @remotable attribute is intended as an alternative to using the @Remotable annotation on the interface

Formatted: Font color: Black

Formatted: Normal, Indent: Before: 54 pt, No bullets or numbering, Don't adjust space between Latin and Asian text

Formatted: Highlight

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

class. The value of the @remotable attribute on the <interface.java/> element does not override the presence of a @Remotable annotation on the interface class and so if the interface class contains a @Remotable annotation and the @remotable attribute has a value of "false", then the SCA Runtime MUST raise an error and MUST NOT run the component concerned. [JCA30005]

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

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

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

Note that the Java interface class identified by the @interface attribute can contain a Java @Callback annotation which identifies a callback interface. If this is the case, then it is not necessary to provide the @callbackInterface attribute. However, 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. [JCA30003]

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

3.2 @Remotable

The ***@Remotable*** annotation on a Java interface, a service implementation class, or a service reference denotes an interface or class that is designed to be used for remote communication. Remotable interfaces are intended to be used for ***coarse grained*** services. Operations' parameters, return values and exceptions are passed ***by-value***. Remotable Services are not allowed to make use of method ***overloading***.

3.3 @Callback

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

3.4 @AsyncInvocation

An interface can be annotated with @AsyncInvocation or with the equivalent @Requires("sca:asyncInvocation") annotation to indicate that request/response operations of that interface are ***long running*** and that response messages are likely to be sent an arbitrary length of time after the initial request message is sent to the target service. This is described in the SCA Assembly Specification [ASSEMBLY].

For a service client, it is strongly recommended that the client uses the asynchronous form of the client interface when using a reference to a service with an interface annotated with @AsyncInvocation, using either polling or callbacks to receive the response message. See the sections "Asynchronous Programming" and the section "JAX-WS Client Asynchronous API for a Synchronous Service" for more details about the asynchronous client API.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

339 For a service implementation, SCA provides an ***asynchronous service*** mapping of the WSDL
340 request/response interface which enables the service implementation to send the response
341 message at an arbitrary time after the original service operation is invoked. This is described in
342 the section "Asynchronous handling of Long Running Service Operations".

343 3.5 SCA Java Annotations for Interface Classes

344 A Java interface referenced by the @interface attribute of an <interface.java/> element MUST NOT
345 contain any of the following SCA Java annotations:

346 @AllowsPassByReference, @ComponentName, @Constructor, @Context, @Destroy, @EagerInit,
347 @Init, @Intent, @Property, @Qualifier, @Reference, @Scope, @Service. [JCA30006]

348 A Java interface referenced by the @callbackInterface attribute of an <interface.java/> element MUST
349 NOT contain any of the following SCA Java annotations:

350 @AllowsPassByReference, @Callback, @ComponentName, @Constructor, @Context, @Destroy,
351 @EagerInit, @Init, @Intent, @Property, @Qualifier, @Reference, @Scope, @Service. [JCA30007]

352 3.6 Compatibility of Java Interfaces

353 The SCA Assembly Model specification [ASSEMBLY] defines a number of criteria that need to be
354 satisfied in order for two interfaces to be compatible or have a compatible superset or subset
355 relationship. If these interfaces are both Java interfaces, compatibility also means that every
356 method that is present in both interfaces is defined consistently in both interfaces with respect to
357 the @OneWay annotation, that is, the annotation is either present in both interfaces or absent in
358 both interfaces. [JCA30009]

Deleted: A Java implementation class referenced by the @interface or the @callbackInterface attribute of an <interface.java/> element MUST NOT contain the following SCA Java annotations:¶
@Intent, @Qualifier. [JCA30008]¶

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

359 # 4 SCA Component Implementation Lifecycle

360 This section describes the lifecycle of an SCA component implementation.

361 ## 4.1 Overview of SCA Component Implementation Lifecycle

362 At a high level, there are 3 main phases through which an SCA component implementation will
363 transition when it is used by an SCA Runtime:

364 1. **The Initialization phase**. This involves constructing an instance of the component
365 implementation class and injecting any properties and references. Once injection is
366 complete, the method annotated with @Init is called, if present, which provides the
367 component implementation an opportunity to perform any internal initialization it requires.

368 2. **The Running phase**. This is where the component implementation has been initialized
369 and the SCA Runtime can dispatch service requests to it over its Service interfaces.

370 3. **The Destroying phase**. This is where the component implementation's scope has ended
371 and the SCA Runtime destroys the component implementation instance. The SCA Runtime
372 calls the method annotated with @Destroy, if present, which provides the component
373 implementation an opportunity to perform any internal clean up that is required.

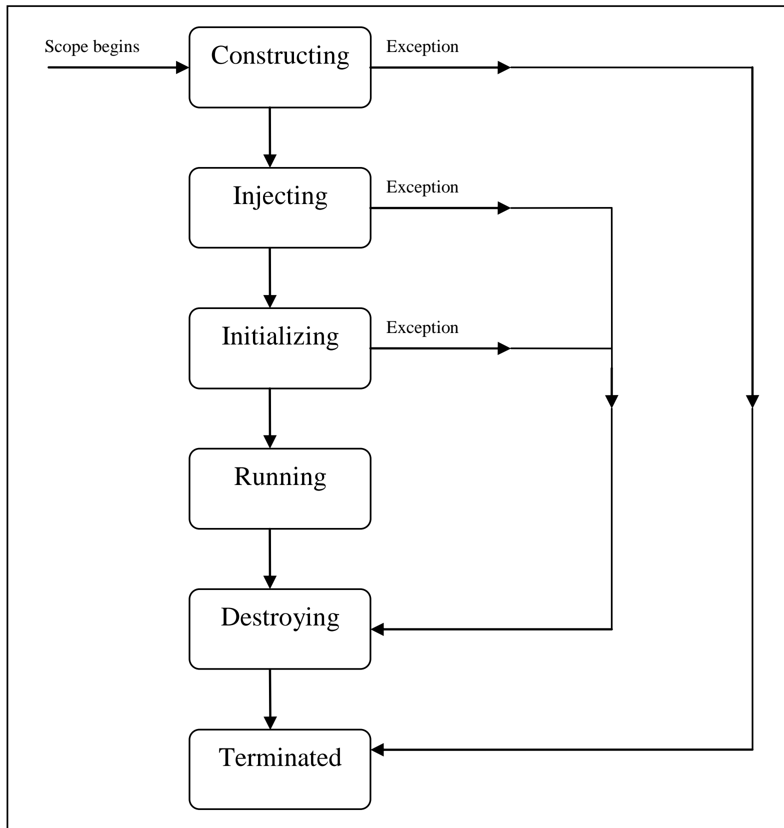
374 ## 4.2 SCA Component Implementation Lifecycle State Diagram

375 The state diagram in Figure 4.1 shows the lifecycle of an SCA component implementation. The
376 sections that follow it describe each of the states that it contains.

377 It should be noted that some component implementation specifications might not implement all
378 states of the lifecycle. In this case, that state of the lifecycle is skipped over.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.



379

380

381 *Figure 4.1 SCA - Component implementation lifecycle*

382 4.2.1 Constructing State

383 The SCA Runtime MUST call a constructor of the component implementation at the start of the
384 Constructing state. [JCA40001] The SCA Runtime MUST perform any constructor reference or
385 property injection when it calls the constructor of a component implementation. [JCA40002]

386 The result of invoking operations on any injected references when the component implementation
387 is in the Constructing state is undefined.

388 When the constructor completes successfully, the SCA Runtime MUST transition the component
389 implementation to the Injecting state. [JCA40003] If an exception is thrown whilst in the
390 Constructing state, the SCA Runtime MUST transition the component implementation to the
391 Terminated state. [JCA40004]

392 4.2.2 Injecting State

393 When a component implementation instance is in the Injecting state, the SCA Runtime MUST first
394 inject all field and setter properties that are present into the component implementation.
395 [JCA40005] The order in which the properties are injected is unspecified.

396 When a component implementation instance is in the Injecting state, the SCA Runtime MUST
397 inject all field and setter references that are present into the component implementation, after all

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

398 the properties have been injected. [JCA40006] The order in which the references are injected is
399 unspecified.

400 The SCA Runtime MUST ensure that the correct synchronization model is used so that all injected
401 properties and references are made visible to the component implementation without requiring the
402 component implementation developer to do any specific synchronization. [JCA40007]

403 The SCA Runtime MUST NOT invoke Service methods on the component implementation when the
404 component implementation is in the Injecting state. [JCA40008]

405 The result of invoking operations on any injected references when the component implementation
406 is in the Injecting state is undefined.

407 When the injection of properties and references completes successfully, the SCA Runtime MUST
408 transition the component implementation to the Initializing state. [JCA40009] If an exception is
409 thrown whilst injecting properties or references, the SCA Runtime MUST transition the component
410 implementation to the Destroying state. [JCA40010] If a property or reference is unable to be
411 injected, the SCA Runtime MUST transition the component implementation to the Destroying
412 state. [JCA40024]

Formatted: Highlight

413 4.2.3 Initializing State

414 When the component implementation enters the Initializing State, the SCA Runtime MUST call the
415 method annotated with @Init on the component implementation, if present. [JCA40011]

416 The component implementation can invoke operations on any injected references when it is in the
417 Initializing state. However, depending on the order in which the component implementations are
418 initialized, the target of the injected reference might not be available since it has not yet been
419 initialized. If a component implementation invokes an operation on an injected reference that
420 refers to a target that has not yet been initialized, the SCA Runtime MUST throw a
421 ServiceUnavailableException. [JCA40012]

422 The SCA Runtime MUST NOT invoke Service methods on the component implementation when the
423 component implementation instance is in the Initializing state. [JCA40013]

424 Once the method annotated with @Init completes successfully, the SCA Runtime MUST transition
425 the component implementation to the Running state. [JCA40014] If an exception is thrown whilst
426 initializing, the SCA Runtime MUST transition the component implementation to the Destroying
427 state. [JCA40015]

428 4.2.4 Running State

429 The SCA Runtime MUST invoke Service methods on a component implementation instance when
430 the component implementation is in the Running state and a client invokes operations on a service
431 offered by the component. [JCA40016]

432 The component implementation can invoke operations on any injected references when the
433 component implementation instance is in the Running state.

434 When the component implementation scope ends, the SCA Runtime MUST transition the
435 component implementation to the Destroying state. [JCA40017]

436 4.2.5 Destroying State

437 When a component implementation enters the Destroying state, the SCA Runtime MUST call the
438 method annotated with @Destroy on the component implementation, if present. [JCA40018]

439 The component implementation can invoke operations on any injected references when it is in the
440 Destroying state. However, depending on the order in which the component implementations are
441 destroyed, the target of the injected reference might no longer be available since it has been
442 destroyed. If a component implementation invokes an operation on an injected reference that
443 refers to a target that has been destroyed, the SCA Runtime MUST throw an
444 InvalidServiceException. [JCA40019]

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

445 The SCA Runtime MUST NOT invoke Service methods on the component implementation when the
446 component implementation instance is in the Destroying state. [JCA40020]

447 Once the method annotated with @Destroy completes successfully, the SCA Runtime MUST
448 transition the component implementation to the Terminated state. [JCA40021] If an exception is
449 thrown whilst destroying, the SCA Runtime MUST transition the component implementation to the
450 Terminated state. [JCA40022]

451 4.2.6 Terminated State

452 The lifecycle of the SCA Component has ended.

453 The SCA Runtime MUST NOT invoke Service methods on the component implementation when the
454 component implementation instance is in the Terminated state. [JCA40023]

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

5 Client API

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

5.1 Accessing Services from an SCA Component

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

5.1.1 Using the Component Context API

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

5.2 Accessing Services from non-SCA Component Implementations

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

5.2.1 SCAClientFactory Interface and Related Classes

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

Objects which implement the SCAClientFactory are obtained using the newInstance() methods of the SCAClientFactory class.

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

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

import java.net.URI;

import org.oasisopen.sca.client.SCAClientFactory;
import org.oasisopen.sca.client.example.HelloService;

/**
 * Example of use of Client API for a client application to obtain
 * an SCA reference proxy for a service in an SCA Domain.
 */
public class Client1 {

   public void someMethod() {

         try {

             String serviceURI = "SomeHelloServiceURI";
             URI domainURI = new URI("SomeDomainURI");

             SCAClientFactory scaClient =
                  SCAClientFactory.newInstance( domainURI );
             HelloService helloService =
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
501                         scaClient.getService(HelloService.class,
502                                                serviceURI);
503                    String reply = helloService.sayHello("Mark");
504
505                } catch (Exception e) {
506                      System.out.println("Received exception");
507                }
508          }
509       }
510
```

511 For details about the SCAClientFactory interface and its related classes see the section
512 "SCAClientFactory Class".

513

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

514 # 6 Error Handling

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

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

518 SCA runtime exceptions are raised by the SCA runtime and signal problems in management of
519 component execution or problems interacting with remote services. The SCA runtime exceptions
520 are defined in the Java API section.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

7 Asynchronous Programming

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

- support for non-blocking method calls
- callbacks

Each of these topics is discussed in the following sections.

7.1 @OneWay

Non-blocking calls represent the simplest form of asynchronous programming, where the client of the service invokes the service and continues processing immediately, without waiting for the service to execute.

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

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

7.2 Callbacks

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

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

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

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

7.2.1 Using Callbacks

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

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

The following example shows a scenario in which bidirectional interfaces and callbacks could be used. A client requests a quotation from a supplier. To process the enquiry and return the quotation, some suppliers might need additional information from the client. The client does not know which additional items of information will be needed by different suppliers. This interaction can be modeled as a bidirectional interface with callback requests to obtain the additional information.

```
package somepackage;
import org.oasisopen.sca.annotation.Callback;
import org.oasisopen.sca.annotation.Remotable;

@Remotable
@Callback(QuotationCallback.class)
public interface Quotation {h
    double requestQuotation(String productCode, int quantity);
}

@Remotable
public interface QuotationCallback {
    String getState();
    String getZipCode();
    String getCreditRating();
}
```

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

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

```

@Callback
protected QuotationCallback callback;

public double requestQuotation(String productCode, int quantity) {
    double price = getPrice(productQuote, quantity);
    double discount = 0;
    if (quantity > 1000 && callback.getState().equals("FL")) {
        discount = 0.05;
    }
    if (quantity > 10000 && callback.getCreditRating().charAt(0) == 'A') {
        discount += 0.05;
    }
    return price * (1-discount);
}
```

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

```

public class ClientImpl implements ClientService, QuotationCallback {

    private QuotationService myService;

```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
620         @Reference
621         public void setMyService(QuotationService service) {
622             myService = service;
623         }
624 
625         public void aClientMethod() {
626             ...
627             double quote = myService.requestQuotation("AB123", 2000);
628             ...
629         }
630 
631         public String getState() {
632             return "TX";
633         }
634         public String getZipCode() {
635             return "78746";
636         }
637         public String getCreditRating() {
638             return "AA";
639         }
640     }
641 
```

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

646 7.2.2 Callback Instance Management

647 Instance management for callback requests received by the client of the bidirectional service is
648 handled in the same way as instance management for regular service requests. If the client
649 implementation has STATELESS scope, the callback is dispatched using a newly initialized
650 instance. If the client implementation has COMPOSITE scope, the callback is dispatched using the
651 same shared instance that is used to dispatch regular service requests.

652 As described in the section "Using Callbacks", a stateful callback can obtain information relating to
653 the original service request from parameters on the callback request. Alternatively, a composite-
654 scoped client could store information relating to the original request as instance data and retrieve
655 it when the callback request is received. These approaches could be combined by using a key
656 passed on the callback request (e.g., an order ID) to retrieve information that was stored in a
657 composite-scoped instance by the client code that made the original request.

658 7.2.3 Callback Injection

659 When a bidirectional service is invoked, the SCA runtime MUST inject a callback reference for the
660 invoking service into all fields and setter methods of the service implementation class that are
661 marked with a @Callback annotation and typed by the callback interface of the bidirectional
662 service, and the SCA runtime MUST inject null into all other fields and setter methods of the
663 service implementation class that are marked with a @Callback annotation. [JCA60001] When a
664 non-bidirectional service is invoked, the SCA runtime MUST inject null into all fields and setter
665 methods of the service implementation class that are marked with a @Callback annotation.
666 [JCA60002]

667 7.2.4 Implementing Multiple Bidirectional Interfaces

668 Since it is possible for a single implementation class to implement multiple services, it is also
669 possible for callbacks to be defined for each of the services that it implements. The service
670 implementation can include an injected field for each of its callbacks. The runtime injects the
671 callback onto the appropriate field based on the type of the callback. The following shows the

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

declaration of two fields, each of which corresponds to a particular service offered by the implementation.

```
@Callback
protected MyService1Callback callback1;

@Callback
protected MyService2Callback callback2;
```

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

7.2.5 Accessing Callbacks

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

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

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

```
@Callback
protected ServiceReference<MyCallback> callback;

public void someMethod() {

   MyCallback myCallback = callback.getService();    …

   myCallback.receiveResult(theResult);
}
```

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

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

```
@Context
ComponentContext context;

public void someMethod() {

   MyCallback myCallback =
         context.getRequestContext().getCallback();

       …

   myCallback.receiveResult(theResult);
}
```

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

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

7.3 Asynchronous handling of Long Running Service Operations

Long-running request-response operations are described in the SCA Assembly Specification [ASSEMBLY]. These operations are characterized by following the WSDL request-response message exchange pattern, but where the timing of the sending of the response message is arbitrarily later than the receipt of the request message, with an impact on the client component, on the service component and also on the transport binding used to communicate between them.

In SCA, such operations are marked with an intent "asyncInvocation" and is expected that the client component, the service component and the binding are all affected by the presence of this intent. This specification does not describe the effects of the intent on the binding, other than to note that in general, there is an implication that the sending of the response message is typically separate from the sending of the request message, typically requiring a separate response endpoint on the client to which the response can be sent.

For components that are clients of a long-running request-response operation, it is strongly recommended that the client makes use of the JAX-WS Client Asynchronous API, either using the polling interface or the callback mechanism described in the section "JAX-WS Client Asynchronous API for a Synchronous Service". The principle is that the client should not synchronously wait for a response from the long running operation since this could take a long time and it is preferable not to tie up resources while waiting.

For the service implementation component, the JAX-WS client asynchronous API is not suitable, so the SCA Java Common Annotations and APIs specification defines the SCA Asynchronous Service interface, which, like the JAX-WS client asynchronous API, is an alternative mapping of a WSDL request-response operation into a Java interface.

7.3.1 SCA Asynchronous Service Interface

The SCA Asynchronous Service interface follows some of the patterns defined by the JAX-WS client asynchronous API, but it is a simpler interface aligned with the needs of a service implementation class.

As an example, for a WSDL portType with a single operation "getPrice" with a String request parameter and a float response, the synchronous Java interface mapping appears in snippet 7-1:

```
// synchronous mapping
public interface StockQuote {
   float getPrice(String ticker);
}
```

Snippet 7-1: Example synchronous Java interface mapping

The JAX-WS client asynchronous API for the same portType adds two asynchronous forms for each synchronous method, as shown in snippet 7-2:

```
// asynchronous mapping
public interface StockQuote {
   float getPrice(String ticker);
   Response<Float> getPriceAsync(String ticker);
   Future<?> getPriceAsync(String ticker, AsyncHandler<Float>);
}
```

Snippet 7-2: Example JAX-WS client asynchronous Java interface mapping

The SCA Asynchronous Service interface has a single method similar to the final one in the asynchronous client interface, as shown in snippet 7-2:

```
// asynchronous mapping
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
@Requires("sca:asyncInvocation")
public interface StockQuote {
    void getPriceAsync(String ticker, ResponseDispatch<Float>);
}
```

Snippet 7-3: Example SCA asynchronous service Java interface mapping

The main characteristics of the SCA asynchronous mapping are:

- there is a single method, with a name with the string "Async" appended to the operation name
- it has a void return type
- it has two input parameters, the first is the request message of the operation and the second is a ResponseDispatch object typed by the response message of the operation (following the rules expressed in the JAX-WS specification for the typing of the AsyncHandler object in the client asynchronous API)
- it is annotated with the asyncInvocation intent
- if the synchronous method has any business faults/exceptions, it is annotated with @AsyncFault, containing a list of the exception classes

Unlike the JAX-WS asynchronous client interface, there is only a single operation for the service implementation to provide (it would be inconvenient for the service implementation to be required to implement multiple methods for each operation in the WSDL interface).

The ResponseDispatch parameter is the mechanism by which the service implementation sends back the response message resulting from the invocation of the service method. The ResponseDispatch is serializable and it can be invoked once at any time after the invocation of the service method, either before or after the service method returns. This enables the service implementation to store the ResponseDispatch in serialized form and release resources while waiting for the completion of whatever activities result from the processing of the initial invocation.

The ResponseDispatch object is allocated by the SCA runtime/binding implementation and it is expected to contain whatever metadata is required to deliver the response message back to the client that invoked the service operation.

The SCA asynchronous service Java interface mapping of a WSDL request-response operation MUST appear as follows:

The interface is annotated with the "asyncInvocation" intent.

For each service operation in the WSDL, the Java interface contains an operation with

- a name which is the JAX-WS mapping of the WSDL operation name, with the suffix "Async" added

- a void return type

- a set of input parameter(s) which match the JAX-WS mapping of the input parameter(s) of the WSDL operation plus an additional last parameter which is a ResponseDispatch object typed by the JAX-WS Response Bean mapping of the output parameter(s) of the WSDL operation, where ResponseDispatch is the type defined in the SCA Java Common Annotations and APIs specification. [JCA60003]

An SCA Runtime MUST support the use of the SCA asynchronous service interface for the interface of an SCA service. [JCA60004]

The ResponseDispatch object passed in as a parameter to a method of a service implementation using the SCA asynchronous service Java interface can be invoked once only through either its sendResponse method or through its sendFault method to return the response resulting from the service method invocation. If the SCA asynchronous

Formatted: Highlight
Formatted: Indent: Before: 36 pt
Deleted: 2
Formatted: Highlight
Deleted: 3
Formatted: Highlight
Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

819 service interface ResponseDispatch handleResponse method is invoked more than once through
820 either its sendResponse or its sendFault method, the SCA runtime MUST throw an
821 IllegalStateException. [JCA60005]

822

823 For the purposes of matching interfaces (when wiring between a reference and a service, or when using
824 an implementation class by a component), an interface which has one or more methods which follow the
825 SCA asynchronous service pattern MUST be treated as if those methods are mapped as the equivalent
826 synchronous methods, as follows:

827 Asynchronous service methods are characterized by:

828 a) void return type

829 b) a method name with the suffix "Async"

830 c) a last input parameter with a type of ResponseDispatch<X>

831 d) annotation with the asyncInvocation intent

832 e) possible annotation with the @AsyncFault annotation

833 The mapping of each such method is as if the method had the return type "X", the method name
834 without the suffix "Async" and all the input parameters except the last parameter of the type
835 ResponseDispatch<X>, plus the list of exceptions contained in the @AsyncFault annotation.
836 [JCA60006]

837

Deleted: 4
Formatted: Highlight
Formatted: Bullets and Numbering
Formatted: Default Paragraph Font, Complex Script Font: 12 pt
Deleted: 5
Deleted: ¶
Formatted: Default Paragraph Font, Font: 18 pt, Complex Script Font: 18 pt
Formatted: Bullets and Numbering

Copyright © OASIS® 2005, 2010. All Rights Reserved.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

8 Policy Annotations for Java

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

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

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

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

This specification supports using the Common Annotations for the Java Platform specification (JSR-250) [JSR-250]. An implication of adopting the common annotation for Java platform specification is that the SCA Java specification supports consistent annotation and Java class inheritance relationships. SCA policy annotation semantics follow the General Guidelines for Inheritance of Annotations in the Common Annotations for the Java Platform specification [JSR-250], except that member-level annotations in a class or interface do not have any effect on how class-level annotations are applied to other members of the class or interface.

8.1 General Intent Annotations

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

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

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

"{" + Namespace URI + "}" + intentname

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

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

```
public static final String SCA_PREFIX =
        "{http://docs.oasis-open.org/ns/opencsa/sca/200912}";
public static final String CONFIDENTIALITY =
        SCA_PREFIX + "confidentiality";
```

Deleted: 200903
Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
    public static final String CONFIDENTIALITY_MESSAGE =
           CONFIDENTIALITY + ".message";
```

Notice that, by convention, qualified intents include the qualifier as part of the name of the constant, separated by an underscore. These intent constants are defined in the file that defines an annotation for the intent (annotations for intents, and the formal definition of these constants, are covered in a following section).

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

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

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

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

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

```
package com.foo;

import static org.oasisopen.sca.annotation.Confidentiality.*;
import static org.oasisopen.sca.annotation.Reference;
import static org.oasisopen.sca.annotation.Requires;

public class Foo {
   @Requires(CONFIDENTIALITY)
   @Reference
   public void setBar(Bar bar) {
      …
   }
}
```

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

```
package com.foo;

import static org.oasisopen.sca.Constants.*;
import static org.oasisopen.sca.annotation.Reference;
import static org.oasisopen.sca.annotation.Requires;

public class Foo {
   @Requires(SCA_PREFIX+"confidentiality")
   @Reference
   public void setBar(Bar bar) {
      …
   }
}
```

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

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

where

QualifiedIntent ::= QName('.' Qualifier)*
Qualifier ::= NCName

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

937 See section @Requires for the formal definition of the @Requires annotation.

8.2 Specific Intent Annotations

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

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

946 For example, the SCA confidentiality intent described in the section on General Intent Annotations
947 using the @Requires(CONFIDENTIALITY) annotation can also be specified with the
948 @Confidentiality specific intent annotation. The specific intent annotation for the "integrity"
949 security intent is:

```
950 @Integrity
```

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

```
952 @Authentication( {"message", "transport"} )
```

953 This annotation attaches the pair of qualified intents: "authentication.message" and
954 "authentication.transport" (the sca: namespace is assumed in this both of these cases –
955 "http://docs.oasis-open.org/ns/opencsa/sca/200912").

Deleted: 200903

956 The general form of specific intent annotations is:

957 '@' Intent ('(' qualifiers ')')?

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

```
959 Intent      ::= NCName
960 qualifiers  ::= '"' qualifier '"' (',"' qualifier '"')*
961 qualifier   ::= NCName ('.' qualifier)?
962
```

8.2.1 How to Create Specific Intent Annotations

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

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

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

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

975 See section @Intent for the formal definition of the @Intent annotation.

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

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

```
982 @Confidentiality({"message","transport"})
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

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

See section @Qualifier for the formal definition of the @Qualifier annotation.

Examples of the use of the @Intent and the @Qualifier annotations in the definition of specific intent annotations are shown in the section dealing with Security Interaction Policy.

8.3 Application of Intent Annotations

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

- Java class
- Java interface
- Method
- Field
- Constructor parameter

Intent annotations MUST NOT be applied to the following:

- A method of a service implementation class, except for a setter method that is either annotated with @Reference or introspected as an SCA reference according to the rules in the appropriate Component Implementation specification
- A service implementation class field that is not either annotated with @Reference or introspected as an SCA reference according to the rules in the appropriate Component Implementation specification
- A service implementation class constructor parameter that is not annotated with @Reference

[JCA70002]

Intent annotations can be applied to classes, interfaces, and interface methods. Applying an intent annotation to a field, setter method, or constructor parameter allows intents to be defined at references. Intent annotations can also be applied to reference interfaces and their methods.

Where multiple intent annotations (general or specific) are applied to the same Java element, the SCA runtime MUST compute the combined intents for the Java element by merging the intents from all intent annotations on the Java element according to the SCA Policy Framework [POLICY] rules for merging intents at the same hierarchy level. [JCA70003]

An example of multiple policy annotations being used together follows:

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

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

If intent annotations are specified on both an interface method and the method's declaring interface, the SCA runtime MUST compute the effective intents for the method by merging the combined intents from the method with the combined intents for the interface according to the SCA Policy Framework [POLICY] rules for merging intents within a structural hierarchy, with the method at the lower level and the interface at the higher level. [JCA70004] This merging process does not remove or change any intents that are applied to the interface.

8.3.1 Intent Annotation Examples

The following examples show how the rules defined in section 8.3 are applied.

Example 8.1 shows how intents on references are merged. In this example, the intents for myRef are "authentication" and "confidentiality.message".

```
@Authentication
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
@Requires(CONFIDENTIALITY)
@Confidentiality("message")
@Reference
protected MyService myRef;
```

Example 8.1. Merging intents on references.

Example 8.2 shows that mutually exclusive intents cannot be applied to the same Java element. In this example, the Java code is in error because of contradictory mutually exclusive intents "managedTransaction" and "noManagedTransaction".

```
@Requires({SCA_PREFIX+"managedTransaction",
           SCA_PREFIX+"noManagedTransaction"})
@Reference
protected MyService myRef;
```

Example 8.2. Mutually exclusive intents.

Example 8.3 shows that intents can be applied to Java service interfaces and their methods. In this example, the effective intents for `MyService.mymethod()` are "authentication" and "confidentiality".

```
@Authentication
public interface MyService {
    @Confidentiality
    public void mymethod();
}
@Service(MyService.class)
public class MyServiceImpl {
    public void mymethod() {...}
}
```

Example 8.3. Intents on Java interfaces, interface methods, and Java classes.

Example 8.4 shows that intents can be applied to Java service implementation classes. In this example, the effective intents for `MyService.mymethod()` are "authentication", "confidentiality", and "managedTransaction".

```
@Authentication
public interface MyService {
    @Confidentiality
    public void mymethod();
}
@Service(MyService.class)
@Requires(SCA_PREFIX+"managedTransaction")
public class MyServiceImpl {
    public void mymethod() {...}
}
```

Example 8.4. Intents on Java service implementation classes.

Example 8.5 shows that intents can be applied to Java reference interfaces and their methods, and also to Java references. In this example, the effective intents for the method `mymethod()` of the reference `myRef` are "authentication", "integrity", and "confidentiality".

```
@Authentication
public interface MyRefInt {
    @Integrity
    public void mymethod();
}
@Service(MyService.class)
public class MyServiceImpl {
    @Confidentiality
    @Reference
    protected MyRefInt myRef;
}
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

Example 8.5. Intents on Java references and their interfaces and methods.

Example 8.6 shows that intents cannot be applied to methods of Java implementation classes. In this example, the Java code is in error because of the @Authentication intent annotation on the implementation method `MyServiceImpl.mymethod()`.

```
public interface MyService {
    public void mymethod();
}
@Service(MyService.class)
public class MyServiceImpl {
    @Authentication
    public void mymethod() {...}
}
```

Example 8.6. Intent on implementation method.

Example 8.7 shows one effect of applying the SCA Policy Framework rules for merging intents within a structural hierarchy to Java service interfaces and their methods. In this example a qualified intent overrides an unqualified intent, so the effective intent for `MyService.mymethod()` is "confidentiality.message".

```
@Confidentiality("message")
public interface MyService {
    @Confidentiality
    public void mymethod();
}
```

Example 8.7. Merging qualified and unqualified intents on Java interfaces and methods.

Example 8.8 shows another effect of applying the SCA Policy Framework rules for merging intents within a structural hierarchy to Java service interfaces and their methods. In this example a lower-level intent causes a mutually exclusive higher-level intent to be ignored, so the effective intent for `mymethod1()` is "managedTransaction" and the effective intent for `mymethod2()` is "noManagedTransaction".

```
@Requires(SCA_PREFIX+"managedTransaction")
public interface MyService {
    public void mymethod1();
    @Requires(SCA_PREFIX+"noManagedTransaction")
    public void mymethod2();
}
```

Example 8.8. Merging mutually exclusive intents on Java interfaces and methods.

8.3.2 Inheritance and Annotation

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

```
package services.hello;
import org.oasisopen.sca.annotation.Authentication;
import org.oasisopen.sca.annotation.Integrity;

@Integrity("transport")
@Authentication
public class HelloService {
      @Integrity
      @Authentication("message")
      public String hello(String message) {...}

      @Integrity
      @Authentication("transport")
      public String helloThere() {...}
}
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```java
package services.hello;
import org.oasisopen.sca.annotation.Authentication;
import org.oasisopen.sca.annotation.Confidentiality;

@Confidentiality("message")
public class HelloChildService extends HelloService {
      @Confidentiality("transport")
      public String hello(String message) {...}
      @Authentication
      String helloWorld() {...}
}
```

Example 8.9. Usage example of annotated policy and inheritance.

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

The effective intent annotation on the ***hello*** method of ***HelloChildService*** is @Confidentiality("transport"),

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

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

Table 8.1 below shows the equivalent declarative security interaction policy of the methods of the HelloService and HelloChildService implementations corresponding to the Java classes shown in Example 8.9.

	Method		
Class	hello()	helloThere()	helloWorld()
HelloService	integrity authentication.message	integrity authentication.transport	N/A
HelloChildService	confidentiality.transport	integrity authentication.transport	authentication confidentiality.message

Table 8.1. Declarative intents equivalent to annotated intents in Example 8.9.

8.4 Relationship of Declarative and Annotated Intents

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

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

8.5 Policy Set Annotations

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

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

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

'@PolicySets({' policySetQName (',' policySetQName)* '})'

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

An example of the @PolicySets annotation:

```
@Reference(name="helloService", required=true)
@PolicySets({ MY_NS + "WS_Encryption_Policy",
              MY_NS + "WS_Authentication_Policy" })
public setHelloService(HelloService service) {
        . . .
}
```

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

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

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

- Java class
- Java interface
- Method
- Field
- Constructor parameter

The @PolicySets annotation MUST NOT be applied to the following:

- A method of a service implementation class, except for a setter method that is either annotated with @Reference or introspected as an SCA reference according to the rules in the appropriate Component Implementation specification
- A service implementation class field that is not either annotated with @Reference or introspected as an SCA reference according to the rules in the appropriate Component Implementation specification
- A service implementation class constructor parameter that is not annotated with @Reference

[JCA70005]

The @PolicySets annotation can be applied to classes, interfaces, and interface methods. Applying a @PolicySets annotation to a field, setter method, or constructor parameter allows policy sets to be defined at references. The @PolicySets annotation can also be applied to reference interfaces and their methods.

If the @PolicySets annotation is specified on both an interface method and the method's declaring interface, the SCA runtime MUST compute the effective policy sets for the method by merging the policy sets from the method with the policy sets from the interface. [JCA70006] This merging process does not remove or change any policy sets that are applied to the interface.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

8.6 Security Policy Annotations

This section introduces annotations for commonly used SCA security intents, as defined in the SCA Policy Framework Specification [POLICY]. Also see the SCA Policy Framework Specification for additional security policy intents that can be used with the @Requires annotation. The following annotations for security policy intents and qualifiers are defined:

- @Authentication
- @Authorization
- @Confidentiality
- @Integrity
- @MutualAuthentication

The @Authentication, @Confidentiality, and @Integrity intents have the same pair of Qualifiers:

- message
- transport

The formal definitions of the security intent annotations are found in the section "Java Annotations".

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

```
package services.hello;
// Interface for HelloService
public interface HelloService {
      String hello(String helloMsg);
}

package services.client;
// Interface for ClientService
public interface ClientService {
      public void clientMethod();
}

// Implementation class for ClientService
package services.client;

import services.hello.HelloService;
import org.oasisopen.sca.annotation.*;

@Service(ClientService.class)
public class ClientServiceImpl implements ClientService {

      private HelloService helloService;

      @Reference(name="helloService", required=true)
      @Integrity("message")
      @Authentication("message")
      public void setHelloService(HelloService service) {
            helloService = service;
      }

      public void clientMethod() {
            String result = helloService.hello("Hello World!");
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

1271 …
1272 }
1273 }

1274

1275 Example 8.10. Usage of security intents on a reference.

1276

8.7 Transaction Policy Annotations

1278 This section introduces annotations for commonly used SCA transaction intents, as defined in the SCA Policy Framework specification [POLICY]. Also see the SCA Policy Framework Specification for additional transaction policy intents that can be used with the @Requires annotation. The following annotations for transaction policy intents and qualifiers are defined:

- @ManagedTransaction
- @NoManagedTransaction
- @SharedManagedTransaction

1285

1286 The @ManagedTransaction intent has the following Qualifiers:

- global
- local

1289

1290 The formal definitions of the transaction intent annotations are found in the section "Java Annotations".

1292 The following example shows an example of applying a transaction intent to a component implementation, where the component implementation requires a global transaction.

1294

```
1295 package services.hello;
1296 // Interface for HelloService
1297 public interface HelloService {
1298       String hello(String helloMsg);
1299 }
1300
1301 // Implementation class for HelloService
1302 package services.hello.impl;
1303
1304 import services.hello.HelloService;
1305 import org.oasisopen.sca.annotation.*;
1306
1307 @Service(HelloService.class)
1308 @ManagedTransaction("global")
1309 public class HelloServiceImpl implements HelloService {
1310
1311       public void someMethod() {
1312             …
1313       }
1314 }
```

1315

1316 Example 8.11. Usage of transaction intents in an implementation.

1317

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

9 Java API

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

9.1 Component Context

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

```
package org.oasisopen.sca;
import java.util.Collection;
public interface ComponentContext {

    String getURI();

    <B> B getService(Class<B> businessInterface, String referenceName);

    <B> ServiceReference<B> getServiceReference(Class<B> businessInterface,
                                                  String referenceName);
    <B> Collection<B> getServices(Class<B> businessInterface,
        String referenceName);

    <B> Collection<ServiceReference<B>> getServiceReferences(Class<B>
        businessInterface, String referenceName);

    <B> ServiceReference<B> createSelfReference(Class<B>
        businessInterface);

    <B> ServiceReference<B> createSelfReference(Class<B> businessInterface,
                                              String serviceName);

    <B> B getProperty(Class<B> type, String propertyName);

    RequestContext getRequestContext();

    <B> ServiceReference<B> cast(B target) throws IllegalArgumentException;

}
```

- ***getURI()*** - returns the absolute URI of the component within the SCA domain
- ***getService(Class<B> businessInterface, String referenceName)*** – Returns a proxy for the reference defined by the current component. The getService() method takes as its input arguments the Java type used to represent the target service on the client and the name of the service reference. It returns an object providing access to the service. The returned object implements the Java interface the service is typed with. The ComponentContext.getService method MUST throw an IllegalArgumentException if the reference identified by the referenceName parameter has multiplicity of 0..n or 1..n.[JCA80001]
- ***getServiceReference(Class<B> businessInterface, String referenceName)*** – Returns a ServiceReference defined by the current component. The returned ServiceReference object implements the interface ***businessInterface***.
    - ***businessinterface –*** the interface class of the reference
    - ***referencename –*** the name of the reference

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

The getServiceReference method MUST throw an IllegalArgumentException if the reference named by the referenceName parameter has multiplicity greater than one. [JCA80004]
The getServiceReference method MUST throw an IllegalArgumentException if the reference named by the referenceName parameter does not have an interface of the type defined by the businessInterface parameter. [JCA80005]
The getServiceReference method MUST throw an IllegalArgumentException if the component does not have a reference with the name provided in the referenceName parameter. [JCA80006]
The getServiceReference method MUST return *null* if the multiplicity of the reference named by the referenceName parameter is 0..1 and the reference has no target service configured. [JCA80007]

- ***getServices(Class<B> businessInterface, String referenceName)*** – Returns a list of typed service proxies for a business interface type and a reference name.
- ***getServiceReferences(Class<B> businessInterface, String referenceName)*** –Returns a list of typed service references for a business interface type and a reference name.
- ***createSelfReference(Class<B> businessInterface)*** – Returns a ServiceReference that can be used to invoke this component over the designated service.
- ***createSelfReference(Class<B> businessInterface, String serviceName)*** – Returns a ServiceReference that can be used to invoke this component over the designated service. The serviceName parameter explicitly declares the service name to invoke
- ***getProperty*** (***Class<B> type, String propertyName***) - Returns the value of an SCA property defined by this component. If a value was specified for the property in the component's SCA configuration, this value is returned; otherwise, null is returned.
- ***getRequestContext()*** - Returns the context for the current SCA service request, or null if there is no current request or if the context is unavailable. 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. [JCA80002]
- ***cast***(***B target***) - Casts a type-safe reference to a ServiceReference

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

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

```
private ComponentContext componentContext;

@Context
public void setContext(ComponentContext context) {
   componentContext = context;
}

public void doSomething() {
   HelloWorld service =
   componentContext.getService(HelloWorld.class,"HelloWorldComponent");
   service.hello("hello");
}
```

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.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

9.2 Request Context

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

```
package org.oasisopen.sca;

import javax.security.auth.Subject;

public interface RequestContext {

   Subject getSecuritySubject();

   String getServiceName();
   <CB> ServiceReference<CB> getCallbackReference();
   <CB> CB getCallback();
   <B> ServiceReference<B> getServiceReference();

}
```

The RequestContext interface has the following methods:

- ***getSecuritySubject()*** – Returns the JAAS Subject of the current request (see the JAAS Reference Guide [JAAS] for details of JAAS)
- ***getServiceName()*** – Returns the name of the service on the Java implementation the request came in on
- ***getCallbackReference()*** – Returns a service reference to the callback as specified by the caller. This method returns null when called for a service request whose interface is not bidirectional or when called for a callback request.
- ***getCallback()*** – Returns a proxy for the callback as specified by the caller. Similar to the getCallbackReference() method, this method returns null when called for a service request whose interface is not bidirectional or when called for a callback request.
- ***getServiceReference()*** – 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. [JCA80003]

9.3 ServiceReference

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

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

```
package org.oasisopen.sca;

public interface ServiceReference<B> extends java.io.Serializable {

    B getService();
    Class<B> getBusinessInterface();
}
```

The ServiceReference interface has the following methods:

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

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

9.4 ResponseDispatch interface

The ***ResponseDispatch*** interface is shown in the following snippet:

```
package org.oasisopen.sca;

public interface ResponseDispatch<T> {
   void sendResponse(T res);
   void sendFault(Throwable e);
   Map<String, Object> getContext();
}
```

Example 9-4: ResponseDispatch interface

sendResponse (T response) method:

Sends the response message from an asynchronous service method. This method can only be invoked once for a given ResponseDispatch object and cannot be invoked if sendFault has previously been invoked for the same ResponseDispatch object.

Returns:

- ***void***

Parameters:

- ***T*** - an instance of the response message returned by the service operation

Exceptions:

- ***InvalidStateexception*** - thrown if either the sendResponse method or the sendFault method has already been called once

sendFault (Throwable e) method:

Sends an exception as a fault from an asynchronous service method. This method can only be invoked once for a given ResponseDispatch object and cannot be invoked if sendResponse has previously been invoked for the same ResponseDispatch object.

Returns:

- ***void***

Parameters:

- ***e*** - an instance of an exception returned by the service operation

Exceptions:

- ***InvalidStateexception*** - thrown if either the sendResponse method or the sendFault method has already been called once

getContext () method:

Obtains the context object for the ResponseDispatch method

Returns:

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

- ***Map<String, object>*** which is the context object for the ResponseDispatch object. The invoker can update the context object with appropriate context information, prior to invoking either the sendResponse method or the sendFault method

Parameters:

- ***none***

Exceptions:

- ***none***

9.5 ServiceRuntimeException

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

```
package org.oasisopen.sca;

public class ServiceRuntimeException extends RuntimeException {
    …
}
```

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

9.6 ServiceUnavailableException

The following snippet shows the ***ServiceUnavailableException***.

```
package org.oasisopen.sca;

public class ServiceUnavailableException extends ServiceRuntimeException {
    …
}
```

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

9.7 InvalidServiceException

The following snippet shows the ***InvalidServiceException***.

```
package org.oasisopen.sca;

public class InvalidServiceException extends ServiceRuntimeException {
    …
}
```

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

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

9.8 Constants

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

```
package org.oasisopen.sca;

public interface Constants {
    String SCA_NS="http://docs.oasis-open.org/ns/opencsa/sca/200912";
    String SCA_PREFIX = "{"+SCA_NS+"}";
}
```

Formatted: Swedish Sweden

Deleted: 200903

9.9 SCAClientFactory Class

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

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

The SCAClientFactory class is as follows:

```

/*
 * 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
 *
 * @author OASIS Open
 */

public abstract class SCAClientFactory {

    protected static SCAClientFactoryFinder factoryFinder;

    private SCAClientFactory() {}

    protected SCAClientFactory(URI domainURI)
```

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
1606          throws NoSuchDomainException {...}
1607
1608       public URI getDomainURI() {...}
1609
1610       public static SCAClientFactory newInstance( URI domainURI )
1611          throws NoSuchDomainException {...}
1612
1613       public static SCAClientFactory newInstance(Properties properties,
1614                                                 URI domainURI)
1615          throws NoSuchDomainException {...}
1616
1617       public static SCAClientFactory newInstance(ClassLoader classLoader,
1618                                                 URI domainURI)
1619          throws NoSuchDomainException {...}
1620
1621       public static SCAClientFactory newInstance(Properties properties,
1622                                          ClassLoader classLoader,
1623                                          URI domainURI)
1624          throws NoSuchDomainException {...}
1625
1626       public abstract <T> T getService(Class<T> interfaze, String serviceURI)
1627             throws NoSuchServiceException, NoSuchDomainException;
1628   }
1629
```

1630 ***newInstance (URI) method:***

1631 Obtains a object implementing the SCAClientFactory class.

1632 Returns:

1633 - ***object*** which implements the SCAClientFactory class

1634 Parameters:

1635–1636 - ***domainURI*** - a URI for the SCA Domain which is targeted by the returned SCAClient object

1637 Exceptions:

1638–1639 - ***NoSuchDomainException*** - thrown if the domainURI parameter does not identify a valid SCA Domain

1640 ***newInstance(Properties, URI) method:***

1641 Obtains a object implementing the SCAClientFactory class, using a specified set of properties.

1642 Returns:

1643 - ***object*** which implements the SCAClientFactory class

1644 Parameters:

1645–1646 - ***properties*** - a set of Properties that can be used when creating the object which implements the SCAClientFactory class.
1647–1648 - ***domainURI*** - a URI for the SCA Domain which is targeted by the returned SCAClient object

1649 Exceptions:

1650–1651 - ***NoSuchDomainException*** - thrown if the domainURI parameter does not identify a valid SCA Domain

1652 ***newInstance(Classloader, URI) method:***

1653 Obtains a object implementing the SCAClientFactory class using a specified classloader.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

1654 Returns:

1655
- ***object*** which implements the SCAClientFactory class

1656 Parameters:

1657
1658
- ***classLoader*** - a ClassLoader to use when creating the object which implements the SCAClientFactory class.

1659
1660
- ***domainURI*** - a URI for the SCA Domain which is targeted by the returned SCAClient object

1661 Exceptions:

1662
1663
- ***NoSuchDomainException*** - thrown if the domainURI parameter does not identify a valid SCA Domain

1664 ***newInstance(Properties, Classloader, URI) method:***

1665
1666 Obtains a object implementing the SCAClientFactory class using a specified set of properties and a specified classloader.

1667 Returns:

1668
- ***object*** which implements the SCAClientFactory class

1669 Parameters:

1670
1671
- ***properties*** - a set of Properties that can be used when creating the object which implements the SCAClientFactory class.

1672
1673
- ***classLoader*** - a ClassLoader to use when creating the object which implements the SCAClientFactory class.

1674
1675
- ***domainURI*** - a URI for the SCA Domain which is targeted by the returned SCAClient object

1676 Exceptions:

1677
1678
- ***NoSuchDomainException*** - thrown if the domainURI parameter does not identify a valid SCA Domain

1679

1680 ***getService method:***

1681 Obtains a proxy reference object for a specified target service in a specified SCA Domain.

1682 Returns:

1683
1684
1685
- ***proxy object*** which implements the business interface T
Invocations of a business method of the proxy causes the invocation of the corresponding operation of the target service .

1686 Parameters:

1687
- ***interfaze*** - a Java interface class which is the business interface of the target service

1688
1689
1690
1691
- ***serviceURI*** - a String containing the relative URI of the target service within its SCA Domain.
Takes the form componentName/serviceName or can also take the extended form componentName/serviceName/bindingName to use a specific binding of the target service

1692 Exceptions:

1693
1694
1695
- ***NoSuchServiceException*** - thrown if a service with the relative URI ***serviceURI*** and a business interface which matches ***interfaze*** cannot be found in the SCA Domain targeted by the SCAClient object

1696
1697
- ***NoSuchDomainException*** - thrown if the domainURI of the SCAClientFactory does not identify a valid SCA Domain

1698

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

1699 ***SCAClientFactory (URI) method:*** a single argument constructor that must be available on all
1700 concrete subclasses of SCAClientFactory. The URI required is the URI of the Domain targeted by
1701 the SCAClientFactory

1702 ***getDomainURI() method:***

1703 Obtains the Domain URI value for this SCAClientFactory

1704 Returns:

1705 - ***URI*** of the target SCA Domain for this SCAClientFactory

1706 Parameters:

1707 - ***none***

1708 Exceptions:

1709 - ***NoSuchDomainException*** - thrown if the domainURI parameter does not identify a valid
1710 SCA Domain

1711

1712 ***private SCAClientFactory() method:*** this private no-argument constructor prevents
1713 instantiation of an SCAClientFactory instance without the use of the constructor with an argument,
1714 even by subclasses of the abstract SCAClientFactory class.

1715 ***factoryFinder protected field:***

1716 Provides a means by which a provider of an SCAClientFactory implementation can inject a factory
1717 finder implementation into the abstract SCAClientFactory class - once this is done, future
1718 invocations of the SCAClientFactory use the injected factory finder to locate and return an instance
1719 of a subclass of SCAClientFactory.

1720

1721 9.10 SCAClientFactoryFinder Interface

1722 The SCAClientFactoryFinder interface is a Service Provider Interface representing a
1723 SCAClientFactory finder. SCA provides a default reference implementation of this interface. SCA
1724 runtime vendors can create alternative implementations of this interface that use different class
1725 loading or lookup mechanisms.

```
1726 package org.oasisopen.sca.client;
1727
1728 public interface SCAClientFactoryFinder {
1729
1730     SCAClientFactory find(Properties properties,
1731                           ClassLoader classLoader,
1732                           URI domainURI )
1733          throws NoSuchDomainException ;
1734 }
```

1735 9.11 SCAClientFactoryFinderImpl Class

1736 This class is a default implementation of an SCAClientFactoryFinder, which is used to find an
1737 implementation of an SCAClientFactory subclass, as used to obtain an SCAClient object for use by
1738 a client. SCA runtime providers can replace this implementation with their own version.

```
1739 package org.oasisopen.sca.client.impl;
1740
1741 public class SCAClientFactoryFinderImpl implements SCAClientFactoryFinder
1742 {
1743    ...
1744     public SCAClientFactoryFinderImpl() {...}
1745
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
1746        public SCAClientFactory find(Properties properties,
1747                                     ClassLoader classLoader
1748                                     URI domainURI)
1749             throws NoSuchDomainException, ServiceRuntimeException {...}
1750       ...
1751    }
```

1752 ***SCAClientFactoryFinderImpl () method:***

1753 Public constructor for the SCAClientFactoryFinderImpl.

1754 Returns:

1755
- ***SCAClientFactoryFinderImpl*** which implements the SCAClientFactoryFinder interface

1756 Parameters:

1757
- ***none***

1758 Exceptions:

1759
- ***none***

1760 ***find (Properties, ClassLoader, URI) method:***

1761 Obtains an implementation of the SCAClientFactory interface. It discovers a provider's
1762 SCAClientFactory implementation by referring to the following information in this order:

1763
1764
1. The org.oasisopen.sca.client.SCAClientFactory property from the Properties specified on the newInstance() method call if specified

1765
2. The org.oasisopen.sca.client.SCAClientFactory property from the System Properties

1766
3. The META-INF/services/org.oasisopen.sca.client.SCAClientFactory file

1767 Returns:

1768
- ***SCAClientFactory*** implementation object

1769 Parameters:

1770
1771
- ***properties*** - a set of Properties that can be used when creating the object which implements the SCAClientFactory interface.

1772
1773
- ***classLoader*** - a ClassLoader to use when creating the object which implements the SCAClientFactory interface.

1774
- ***domainURI*** - a URI for the SCA Domain targeted by the SCAClientFactory

1775 Exceptions:

1776
- ***ServiceRuntimeException*** - if the SCAClientFactory implementation could not be found

1777 9.12 NoSuchDomainException

1778 The following shows the NoSuchDomainException:

```
1779    package org.oasisopen.sca;
1780
1781    public class NoSuchDomainException extends Exception {
1782       ...
1783    }
```

1784 This exception indicates that the Domain specified could not be found.

1785 9.13 NoSuchServiceException

1786 The following shows the NoSuchServiceException:

```
1787    package org.oasisopen.sca;
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

1788
1789
1790
1791

```
public class NoSuchServiceException extends Exception {
    ...
}
```

1792 This exception indicates that the service specified could not be found.

1793

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

10 Java Annotations

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

This specification places constraints on some annotations that are not detectable by a Java compiler. For example, the definition of the @Property and @Reference annotations indicate that they are allowed on parameters, but the sections "@Property" and "@Reference" constrain those definitions to constructor parameters. 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. [JCA90001]

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. [JCA90002]

10.1 @AllowsPassByReference

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

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.ElementType.TYPE;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Target({TYPE, METHOD, FIELD, PARAMETER})
@Retention(RUNTIME)
public @interface AllowsPassByReference {

   boolean value() default true;
}
```

The @AllowsPassByReference annotation allows service method implementations and client references to be marked as "allows pass by reference" to indicate that they use input parameters, return values and exceptions in a manner that allows the SCA runtime to avoid the cost of copying mutable objects when a remotable service is called locally within the same JVM.

The @AllowsPassByReference annotation has the following attribute:

- ***value*** – specifies whether the "allows pass by reference" marker applies to the service implementation class, service implementation method, or client reference to which this annotation applies; if not specified, defaults to true.

The @AllowsPassByReference annotation MUST only annotate the following locations:

a service implementation class

an individual method of a remotable service implementation

- an individual reference which uses a remotable interface, where the reference is a field, a setter method, or a constructor parameter [JCA90052]

The "allows pass by reference" marking of a method implementation of a remotable service is determined as follows:

1. If the method has an @AllowsPassByReference annotation, the method is marked "allows pass by reference" if and only if the value of the method's annotation is true.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

2. Otheriwse, if the class has an @AllowsPassByReference annotation, the method is marked "allows pass by reference" if and only if the value of the class's annotation is true.
3. Otherwise, the method is not marked "allows pass by reference".

The "allows pass by reference" marking of a reference for a remotable service is determined as follows:

1. If the reference has an @AllowsPassByReference annotation, the reference is marked "allows pass by reference" if and only if the value of the reference's annotation is true.
2. Otherwise, if the service implementation class containing the reference has an @AllowsPassByReference annotation, the reference is marked "allows pass by reference" if and only if the value of the class's annotation is true.
3. Otherwise, the reference is not marked "allows pass by reference".

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

```
@AllowsPassByReference
public String hello(String message) {
    …
}
```

The following snippet shows a sample where @AllowsPassByReference is defined for a client reference of a Java component implementation class.

```
@AllowsPassByReference
@Reference
private StockQuoteService stockQuote;
```

10.2 @AsyncFault

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

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import static org.oasisopen.sca.Constants.SCA_PREFIX;

import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Inherited
@Target({METHOD})
@Retention(RUNTIME)
public @interface AsyncInvocation {

    Class<?>[] value() default {};

}
```

The ***@AsyncInvocation*** annotation is used to indicate the faults/exceptions which are returned by the asynchronous service method which it annotates.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

10.3 @AsyncInvocation

The following Java code defines the ***@AsyncInvocation*** annotation, which is used to attach the "asyncInvocation" policy intent to an interface or to a method:

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import static org.oasisopen.sca.Constants.SCA_PREFIX;

import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Inherited
@Target({TYPE, METHOD})
@Retention(RUNTIME)
@Intent(AsyncInvocation.ASYNCINVOCATION)
public @interface AsyncInvocation {
    String ASYNCINVOCATION = SCA_PREFIX + "asyncInvocation";

    boolean value() default true;
}
```

The ***@AsyncInvocation*** annotation is used to indicate that the operations of a Java interface uses the long-running request-response pattern as described in the SCA Assembly specification.

10.4 @Authentication

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

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import static org.oasisopen.sca.Constants.SCA_PREFIX;

import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Inherited
@Target({TYPE, FIELD, METHOD, PARAMETER})
@Retention(RUNTIME)
@Intent(Authentication.AUTHENTICATION)
public @interface Authentication {
    String AUTHENTICATION = SCA_PREFIX + "authentication";
    String AUTHENTICATION_MESSAGE = AUTHENTICATION + ".message";
    String AUTHENTICATION_TRANSPORT = AUTHENTICATION + ".transport";

    /**
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
    * List of authentication qualifiers (such as "message"
    * or "transport").
    *
    * @return authentication qualifiers
    */
   @Qualifier
   String[] value() default "";
}
```

The **@Authentication** annotation is used to indicate the need for authentication. See the SCA Policy Framework Specification [POLICY] for details on the meaning of the intent. See the section on Application of Intent Annotations for samples of how intent annotations are used in Java.

10.5 @Authorization

The following Java code defines the @Authorization annotation:

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import static org.oasisopen.sca.Constants.SCA_PREFIX;

import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

/**
 * The @Authorization annotation is used to indicate that
 * an authorization policy is required.
 */
@Inherited
@Target({TYPE, FIELD, METHOD, PARAMETER})
@Retention(RUNTIME)
@Intent(Authorization.AUTHORIZATION)
public @interface Authorization {
    String AUTHORIZATION = SCA_PREFIX + "authorization";
}
```

The **@Authorization** annotation is used to indicate the need for an authorization policy. See the SCA Policy Framework Specification [POLICY] for details on the meaning of the intent. See the section on Application of Intent Annotations for samples of how intent annotations are used in Java.

10.6 @Callback

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

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
```


Copyright © OASIS® 2005, 2010. All Rights Reserved.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

```
import java.lang.annotation.Target;

@Target({TYPE, METHOD, FIELD})
@Retention(RUNTIME)
public @interface Callback {

   Class<?> value() default Void.class;
}
```

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

The @Callback annotation has the following attribute:

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

The @Callback annotation can also be used to annotate a method or a field of an SCA implementation class, in order to have a callback object injected. 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. [JCA90046] When used to annotate a method or a field of an implementation class for injection of a callback object, the type of the method or field MUST be the callback interface of at least one bidirectional service offered by the implementation class. [JCA90054] When used to annotate a setter method or a field of an implementation class for injection of a callback object, the SCA runtime MUST inject a callback reference proxy into that method or field when the Java class is initialized, if the component is invoked via a service which has a callback interface and where the type of the setter method or field corresponds to the type of the callback interface. [JCA90058]

The @Callback annotation MUST NOT appear on a setter method or a field of a Java implementation class that has COMPOSITE scope. [JCA90057]

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

```
package somepackage;
import org.oasisopen.sca.annotation.Callback;
import org.oasisopen.sca.annotation.Remotable;
@Remotable
@Callback(MyServiceCallback.class)
public interface MyService {

    void someMethod(String arg);
}

@Remotable
public interface MyServiceCallback {

    void receiveResult(String result);
}
```

In this example, the implied component type is:

```
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" >

   <service name="MyService">
         <interface.java interface="somepackage.MyService"
                  callbackInterface="somepackage.MyServiceCallback"/>
   </service>
</componentType>
```

Deleted: 200903
Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

10.7 @ComponentName

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

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.TYPE;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface ComponentName {

}
```

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

The following snippet shows a component name field definition sample.

```
@ComponentName
private String componentName;
```

The following snippet shows a component name setter method sample.

```
@ComponentName
public void setComponentName(String name) {
 //…
}
```

10.8 @Confidentiality

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

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import static org.oasisopen.sca.Constants.SCA_PREFIX;

import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Inherited
@Target({TYPE, FIELD, METHOD, PARAMETER})
@Retention(RUNTIME)
@Intent(Confidentiality.CONFIDENTIALITY)
public @interface Confidentiality {
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
2100        String CONFIDENTIALITY = SCA_PREFIX + "confidentiality";
2101        String CONFIDENTIALITY_MESSAGE = CONFIDENTIALITY + ".message";
2102        String CONFIDENTIALITY_TRANSPORT = CONFIDENTIALITY + ".transport";
2103
2104        /**
2105         * List of confidentiality qualifiers such as "message" or
2106         * "transport".
2107         *
2108         * @return confidentiality qualifiers
2109         */
2110        @Qualifier
2111        String[] value() default "";
2112    }
```

2113 The ***@Confidentiality*** annotation is used to indicate the need for confidentiality. See the SCA Policy
2114 Framework Specification [POLICY] for details on the meaning of the intent. See the section on
2115 Application of Intent Annotations for samples of how intent annotations are used in Java.

2116 10.9 @Constructor

2117 The following Java code defines the ***@Constructor*** annotation:

```
2118
2119    package org.oasisopen.sca.annotation;
2120
2121    import static java.lang.annotation.ElementType.CONSTRUCTOR;
2122    import static java.lang.annotation.RetentionPolicy.RUNTIME;
2123    import java.lang.annotation.Retention;
2124    import java.lang.annotation.Target;
2125
2126    @Target(CONSTRUCTOR)
2127    @Retention(RUNTIME)
2128    public @interface Constructor { }
2129
```

2130 The @Constructor annotation is used to mark a particular constructor to use when instantiating a
2131 Java component implementation. If a constructor of an implementation class is annotated with
2132 @Constructor and the constructor has parameters, each of these parameters MUST have either a
2133 @Property annotation or a @Reference annotation. [JCA90003]

2134 The following snippet shows a sample for the @Constructor annotation.

```
2135
2136    public class HelloServiceImpl implements HelloService {
2137
2138       public HelloServiceImpl(){
2139        ...
2140        }
2141
2142        @Constructor
2143        public HelloServiceImpl(@Property(name="someProperty")
2144                                String someProperty ){
2145        ...
2146        }
2147
2148         public String hello(String message) {
2149             ...
2150          }
2151    }
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

10.10 @Context

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

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface Context {

}
```

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

The @Context annotation has no attributes.

The following snippet shows a ComponentContext field definition sample.

```
@Context
protected ComponentContext context;
```

The following snippet shows a RequestContext field definition sample.

```
@Context
protected RequestContext context;
```

10.11 @Destroy

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

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Target(METHOD)
@Retention(RUNTIME)
public @interface Destroy {

}
```


Copyright © OASIS® 2005, 2010. All Rights Reserved.

2199 The @Destroy annotation is used to denote a single Java class method that will be called when the
2200 scope defined for the implementation class ends. A method annotated with @Destroy can have
2201 any access modifier and MUST have a void return type and no arguments. [JCA90004]

2202 If there is a method annotated with @Destroy that matches the criteria for the annotation, the
2203 SCA runtime MUST call the annotated method when the scope defined for the implementation
2204 class ends. [JCA90005]

2205 The following snippet shows a sample for a destroy method definition.

2206

```
2207 @Destroy
2208 public void myDestroyMethod() {
2209     …
2210 }
```

2211 10.12 @EagerInit

2212 The following Java code defines the ***@EagerInit*** annotation:

2213

```
2214 package org.oasisopen.sca.annotation;
2215
2216 import static java.lang.annotation.ElementType.TYPE;
2217 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2218 import java.lang.annotation.Retention;
2219 import java.lang.annotation.Target;
2220
2221 @Target(TYPE)
2222 @Retention(RUNTIME)
2223 public @interface EagerInit {
2224
2225 }
2226
```

2227 The ***@EagerInit*** annotation is used to mark the Java class of a COMPOSITE scoped
2228 implementation for eager initialization. When marked for eager initialization with an @EagerInit
2229 annotation, the composite scoped instance MUST be created when its containing component is
2230 started. [JCA90007]

2231 10.13 @Init

2232 The following Java code defines the ***@Init*** annotation:

2233

```
2234 package org.oasisopen.sca.annotation;
2235
2236 import static java.lang.annotation.ElementType.METHOD;
2237 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2238 import java.lang.annotation.Retention;
2239 import java.lang.annotation.Target;
2240
2241 @Target(METHOD)
2242 @Retention(RUNTIME)
2243 public @interface Init {
2244
2245
2246 }
2247
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

2248 The @Init annotation is used to denote a single Java class method that is called when the scope
2249 defined for the implementation class starts. A method marked with the @Init annotation can have
2250 any access modifier and MUST have a void return type and no arguments. [JCA90008]

2251 If there is a method annotated with @Init that matches the criteria for the annotation, the SCA
2252 runtime MUST call the annotated method after all property and reference injection is complete.
2253 [JCA90009]

2254 The following snippet shows an example of an init method definition.

2255

```
2256 @Init
2257 public void myInitMethod() {
2258     …
2259 }
```

2260 10.14 @Integrity

2261 The following Java code defines the ***@Integrity*** annotation:

```
2262
2263 package org.oasisopen.sca.annotation;
2264
2265 import static java.lang.annotation.ElementType.FIELD;
2266 import static java.lang.annotation.ElementType.METHOD;
2267 import static java.lang.annotation.ElementType.PARAMETER;
2268 import static java.lang.annotation.ElementType.TYPE;
2269 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2270 import static org.oasisopen.sca.Constants.SCA_PREFIX;
2271
2272 import java.lang.annotation.Inherited;
2273 import java.lang.annotation.Retention;
2274 import java.lang.annotation.Target;
2275
2276 @Inherited
2277 @Target({TYPE, FIELD, METHOD, PARAMETER})
2278 @Retention(RUNTIME)
2279 @Intent(Integrity.INTEGRITY)
2280 public @interface Integrity {
2281     String INTEGRITY = SCA_PREFIX + "integrity";
2282     String INTEGRITY_MESSAGE = INTEGRITY + ".message";
2283     String INTEGRITY_TRANSPORT = INTEGRITY + ".transport";
2284
2285     /**
2286      * List of integrity qualifiers (such as "message" or "transport").
2287      *
2288      * @return integrity qualifiers
2289      */
2290     @Qualifier
2291     String[] value() default "";
2292 }
2293
```

2294 The ***@Integrity*** annotation is used to indicate that the invocation requires integrity (i.e. no
2295 tampering of the messages between client and service). See the SCA Policy Framework
2296 Specification [POLICY] for details on the meaning of the intent. See the section on Application of
2297 Intent Annotations for samples of how intent annotations are used in Java.

2298 10.15 @Intent

2299 The following Java code defines the ***@Intent*** annotation:

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Target({ANNOTATION_TYPE})
@Retention(RUNTIME)
public @interface Intent {
    /**
     * The qualified name of the intent, in the form defined by
     * {@link javax.xml.namespace.QName#toString}.
     * @return the qualified name of the intent
     */
    String value() default "";

    /**
     * The XML namespace for the intent.
     * @return the XML namespace for the intent
     */
    String targetNamespace() default "";

    /**
     * The name of the intent within its namespace.
     * @return name of the intent within its namespace
     */
    String localPart() default "";
}
```

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

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

10.16 @ManagedSharedTransaction

The following Java code defines the @ManagedSharedTransaction annotation:

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import static org.oasisopen.sca.Constants.SCA_PREFIX;

import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

/**
 * The @ManagedSharedTransaction annotation is used to indicate that
 * a distributed ACID transaction is required.
 */
@Inherited
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

2355
2356
2357
2358
2359
2360
2361

```
@Target({TYPE, FIELD, METHOD, PARAMETER})
@Retention(RUNTIME)
@Intent(ManagedSharedTransaction.MANAGEDSHAREDTRANSACTION)
public @interface ManagedSharedTransaction {
    String MANAGEDSHAREDTRANSACTION = SCA_PREFIX +
"managedSharedTransaction";
}
```

2362 The ***@ManagedSharedTransaction*** annotation is used to indicate the need for a distributed and
2363 globally coordinated ACID transaction. See the SCA Policy Framework Specification [POLICY] for
2364 details on the meaning of the intent. See the section on Application of Intent Annotations for
2365 samples of how intent annotations are used in Java.

2366 10.17 @ManagedTransaction

2367 The following Java code defines the @ManagedTransaction annotation:

2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400

```
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import static org.oasisopen.sca.Constants.SCA_PREFIX;

import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

/**
 * The @ManagedTransaction annotation is used to indicate the
 * need for an ACID transaction environment.
 */
@Inherited
@Target({TYPE, FIELD, METHOD, PARAMETER})
@Retention(RUNTIME)
@Intent(ManagedTransaction.MANAGEDTRANSACTION)
public @interface ManagedTransaction {
    String MANAGEDTRANSACTION = SCA_PREFIX + "managedTransaction";
    String MANAGEDTRANSACTION_MESSAGE = MANAGEDTRANSACTION + ".local";
    String MANAGEDTRANSACTION_TRANSPORT = MANAGEDTRANSACTION + ".global";

    /**
     * List of managedTransaction qualifiers (such as "global" or
"local").
     *
     * @return managedTransaction qualifiers
     */
    @Qualifier
    String[] value() default "";
}
```

2401 The ***@ManagedTransaction*** annotation is used to indicate the need for an ACID transaction. See
2402 the SCA Policy Framework Specification [POLICY] for details on the meaning of the intent. See the
2403 section on Application of Intent Annotations for samples of how intent annotations are used in
2404 Java.

2405 10.18 @MutualAuthentication

2406 The following Java code defines the @MutualAuthentication annotation:

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
2407    package org.oasisopen.sca.annotation;
2408
2409    import static java.lang.annotation.ElementType.FIELD;
2410    import static java.lang.annotation.ElementType.METHOD;
2411    import static java.lang.annotation.ElementType.PARAMETER;
2412    import static java.lang.annotation.ElementType.TYPE;
2413    import static java.lang.annotation.RetentionPolicy.RUNTIME;
2414    import static org.oasisopen.sca.Constants.SCA_PREFIX;
2415
2416    import java.lang.annotation.Inherited;
2417    import java.lang.annotation.Retention;
2418    import java.lang.annotation.Target;
2419
2420    /**
2421     * The @MutualAuthentication annotation is used to indicate that
2422     * a mutual authentication policy is needed.
2423     */
2424    @Inherited
2425    @Target({TYPE, FIELD, METHOD, PARAMETER})
2426    @Retention(RUNTIME)
2427    @Intent(MutualAuthentication.MUTUALAUTHENTICATION)
2428    public @interface MutualAuthentication {
2429        String MUTUALAUTHENTICATION = SCA_PREFIX + "mutualAuthentication";
2430    }
```

2431 The ***@MutualAuthentication*** annotation is used to indicate the need for mutual authentication
2432 between a service consumer and a service provider. See the SCA Policy Framework Specification
2433 [POLICY] for details on the meaning of the intent. See the section on Application of Intent
2434 Annotations for samples of how intent annotations are used in Java.

2435 10.19 @NoManagedTransaction

2436 The following Java code defines the @NoManagedTransaction annotation:

```
2437    package org.oasisopen.sca.annotation;
2438
2439    import static java.lang.annotation.ElementType.FIELD;
2440    import static java.lang.annotation.ElementType.METHOD;
2441    import static java.lang.annotation.ElementType.PARAMETER;
2442    import static java.lang.annotation.ElementType.TYPE;
2443    import static java.lang.annotation.RetentionPolicy.RUNTIME;
2444    import static org.oasisopen.sca.Constants.SCA_PREFIX;
2445
2446    import java.lang.annotation.Inherited;
2447    import java.lang.annotation.Retention;
2448    import java.lang.annotation.Target;
2449
2450    /**
2451     * The @NoManagedTransaction annotation is used to indicate that
2452     * a non-transactional environment is needed.
2453     */
2454    @Inherited
2455    @Target({TYPE, FIELD, METHOD, PARAMETER})
2456    @Retention(RUNTIME)
2457    @Intent(NoManagedTransaction.NOMANAGEDTRANSACTION)
2458    public @interface NoManagedTransaction {
2459        String NOMANAGEDTRANSACTION = SCA_PREFIX + "noManagedTransaction";
2460    }
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

2461 The ***@NoManagedTransaction*** annotation is used to indicate that the component does not want
2462 to run in an ACID transaction. See the SCA Policy Framework Specification [POLICY] for details on
2463 the meaning of the intent. See the section on Application of Intent Annotations for samples of how
2464 intent annotations are used in Java.

2465 10.20 @OneWay

2466 The following Java code defines the ***@OneWay*** annotation:

2467

```
2468 package org.oasisopen.sca.annotation;
2469
2470 import static java.lang.annotation.ElementType.METHOD;
2471 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2472 import java.lang.annotation.Retention;
2473 import java.lang.annotation.Target;
2474
2475 @Target(METHOD)
2476 @Retention(RUNTIME)
2477 public @interface OneWay {
2478
2479
2480 }
2481
```

2482 A method annotated with @OneWay MUST have a void return type and MUST NOT have declared
2483 checked exceptions. [JCA90055]

2484 When a method of a Java interface is annotated with @OneWay, the SCA runtime MUST ensure
2485 that all invocations of that method are executed in a non-blocking fashion, as described in the
2486 section on Asynchronous Programming. [JCA90056]

2487 The @OneWay annotation has no attributes.

2488 The following snippet shows the use of the @OneWay annotation on an interface.

```
2489 package services.hello;
2490
2491 import org.oasisopen.sca.annotation.OneWay;
2492
2493 public interface HelloService {
2494     @OneWay
2495     void hello(String name);
2496 }
```

2497 10.21 @PolicySets

2498 The following Java code defines the ***@PolicySets*** annotation:

```
2499
2500 package org.oasisopen.sca.annotation;
2501
2502 import static java.lang.annotation.ElementType.FIELD;
2503 import static java.lang.annotation.ElementType.METHOD;
2504 import static java.lang.annotation.ElementType.PARAMETER;
2505 import static java.lang.annotation.ElementType.TYPE;
2506 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2507
2508 import java.lang.annotation.Retention;
2509 import java.lang.annotation.Target;
2510
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
@Target({TYPE, FIELD, METHOD, PARAMETER})
@Retention(RUNTIME)
public @interface PolicySets {
    /**
     * Returns the policy sets to be applied.
     *
     * @return the policy sets to be applied
     */
    String[] value() default "";
}
```

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

See the section "Policy Set Annotations" for details and samples.

10.22 @Property

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

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Target({METHOD, FIELD, PARAMETER})
@Retention(RUNTIME)
public @interface Property {

   String name() default "";
   boolean required() default true;
}
```

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

When the Java type of a field, setter method or constructor parameter with the @Property annotation is a primitive type or a JAXB annotated class, the SCA runtime MUST convert a property value specified by an SCA component definition into an instance of the Java type as defined by the XML to Java mapping in the JAXB specification [JAXB] with XML schema validation enabled. [JCA90061]

When the Java type of a field, setter method or constructor parameter with the @Property annotation is not a JAXB annotated class, the SCA runtime can use any XML to Java mapping when converting property values into instances of the Java type.

The @Property annotation MUST NOT be used on a class field that is declared as final. [JCA90011]

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

The @Property annotation has the following attributes:

- ***name (optional)*** – the name of the property. For a field annotation, the default is the name of the field of the Java class. For a setter method annotation, the default is the JavaBeans property name [JAVABEANS] corresponding to the setter method name. For a

Formatted: Highlight

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

@Property annotation applied to a constructor parameter, there is no default value for the name attribute and the name attribute MUST be present. [JCA90013]

- ***required (optional)*** – a boolean value which specifies whether injection of the property value is required or not, where true means injection is required and false means injection is not required. Defaults to true. For a @Property annotation applied to a constructor parameter, the required attribute MUST NOT have the value false. [JCA90014]

Deleted: For a @Property annotation applied to a constructor parameter, the required attribute MUST have the value true.

The following snippet shows a property field definition sample.

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

The following snippet shows a property setter sample

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

For a @Property annotation, if 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.[JCA90047]

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

```
...
private List<String> helloConfigurationProperty;

@Property(required=true)
public void setHelloConfigurationProperty(List<String> property) {
        helloConfigurationProperty = property;
}
...
```

10.23 @Qualifier

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

```

package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Target(METHOD)
@Retention(RUNTIME)
public @interface Qualifier {
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
}
```

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

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

10.24 @Reference

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

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
@Target({METHOD, FIELD, PARAMETER})
@Retention(RUNTIME)
public @interface Reference {

   String name() default "";
   boolean required() default true;
}
```

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

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

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

The @Reference annotation has the following attributes:

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

The following snippet shows a reference field definition sample.

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

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

The following snippet shows a reference setter sample

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

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.

```
package services.hello;

private HelloService helloService;

@Reference(name="helloService", required=true)
public setHelloService(HelloService service) {
         helloService = service;
}

public void clientMethod() {
         String result = helloService.hello("Hello World!");
         …
}
```

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.

```
<?xml version="1.0" encoding="ASCII"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">

   <!-- Any services offered by the component would be listed here -->
   <reference name="helloService" multiplicity="1..1">
         <interface.java interface="services.hello.HelloService"/>
   </reference>

</componentType>
```

Deleted: 200903

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. [JCA90020]

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. [JCA90021]

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

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

```
    @Reference(name="helloServices", required=true)
    protected List<HelloService> helloServices;

    public void clientMethod() {

          …
          for (int index = 0; index < helloServices.size(); index++) {
                HelloService helloService =
                      (HelloService)helloServices.get(index);
                String result = helloService.hello("Hello World!");
          }
          …
    }
```

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

```
<?xml version="1.0" encoding="ASCII"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">

    <!-- Any services offered by the component would be listed here -->
    <reference name="helloServices" multiplicity="1..n">
          <interface.java interface="services.hello.HelloService"/>
    </reference>

</componentType>
```

Deleted: 200903

An unwired reference with a multiplicity of 0..1 MUST be presented to the implementation code by the SCA runtime as null [JCA90022] 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 [JCA90023]

10.24.1 Reinjection

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. [JCA90024]

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.

[JCA90025]

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

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. [JCA90026]

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

2760 If an operation is called on a reference where the target of that reference has been undeployed,
2761 the SCA runtime SHOULD throw an InvalidServiceException. [JCA90027] If an operation is called
2762 on a reference where the target of the reference has become unavailable for some reason, the
2763 SCA runtime SHOULD throw a ServiceUnavailableException. [JCA90028] If the target service of
2764 the reference is changed, the reference MUST either continue to work or throw an
2765 InvalidServiceException when it is invoked. [JCA90029] If it doesn't work, the exception thrown
2766 will depend on the runtime and the cause of the failure.

2767 A ServiceReference that has been obtained from a reference by ComponentContext.cast()
2768 corresponds to the reference that is passed as a parameter to cast(). If the reference is
2769 subsequently reinjected, the ServiceReference obtained from the original reference MUST continue
2770 to work as if the reference target was not changed. [JCA90030] If the target of a ServiceReference
2771 has been undeployed, the SCA runtime SHOULD throw a InvalidServiceException when an
2772 operation is invoked on the ServiceReference. [JCA90031] If the target of a ServiceReference has
2773 become unavailable, the SCA runtime SHOULD throw a ServiceUnavailableException when an
2774 operation is invoked on the ServiceReference. [JCA90032] If the target service of a
2775 ServiceReference is changed, the reference MUST either continue to work or throw an
2776 InvalidServiceException when it is invoked. [JCA90033] If it doesn't work, the exception thrown
2777 will depend on the runtime and the cause of the failure.

2778 A reference or ServiceReference accessed through the component context by calling getService()
2779 or getServiceReference() MUST correspond to the current configuration of the domain. This applies
2780 whether or not reinjection has taken place. [JCA90034] If the target of a reference or
2781 ServiceReference accessed through the component context by calling getService() or
2782 getServiceReference() has been undeployed or has become unavailable, the result SHOULD be a
2783 reference to the undeployed or unavailable service, and attempts to call business methods
2784 SHOULD throw an InvalidServiceException or a ServiceUnavailableException. [JCA90035] If the
2785 target service of a reference or ServiceReference accessed through the component context by
2786 calling getService() or getServiceReference() has changed, the returned value SHOULD be a
2787 reference to the changed service. [JCA90036]

2788 The rules for reference reinjection also apply to references with a multiplicity of 0..n or 1..n. This
2789 means that in the cases where reference reinjection is not allowed, the array or Collection for a
2790 reference of multiplicity 0..n or multiplicity 1..n MUST NOT change its contents when changes
2791 occur to the reference wiring or to the targets of the wiring. [JCA90037] In cases where the
2792 contents of a reference array or collection change when the wiring changes or the targets change,
2793 then for references that use setter injection, the setter method MUST be called by the SCA
2794 runtime for any change to the contents. [JCA90038] A reinjected array or Collection for a
2795 reference MUST NOT be the same array or Collection object previously injected to the component.
2796 [JCA90039]

2797

	Effect on		
Change event	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 continues to work as if the reference target was not changed.	continue to work as if the reference target was not changed.	Result corresponds to the current configuration of the domain.
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	Business methods throw ServiceUnavailableExce	Business methods throw ServiceUnavailableExce	Result is be a reference to the unavailable service. Business

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

becomes unavailable	ption	ption	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.
* Other conditions: The component cannot be STATELESS scoped. The reference has to use either field-based injection or setter injection. References that are injected through constructor injection cannot be changed. ** Result of invoking ComponentContext.cast() corresponds to the reference that is passed as a parameter to cast().			

2798

10.25 @Remotable

2799

2800 The following Java code defines the ***@Remotable*** annotation:

2801

```
2802 package org.oasisopen.sca.annotation;
2803
2804 import static java.lang.annotation.ElementType.TYPE;
2805 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2806 import java.lang.annotation.Retention;
2807 import java.lang.annotation.Target;
2808
2809
2810 @Target(TYPE)
2811 @Retention(RUNTIME)
2812 public @interface Remotable {
2813
2814 }
2815
```

2816 The @Remotable annotation is used to indicate that an SCA service interface is remotable. The
2817 @Remotable annotation is valid only on a Java interface, a Java class, a field, a setter method, or
2818 a constructor parameter. It MUST NOT appear anywhere else. [JCA90053] A remotable service
2819 can be published externally as a service and MUST be translatable into a WSDL portType.
2820 [JCA90040]

2821 The @Remotable annotation has no attributes. When placed on a Java service interface, it
2822 indicates that the interface is remotable. When placed on a Java service implementation class, it
2823 indicates that all SCA service interfaces provided by the class (including the class itself, if the class
2824 defines an SCA service interface) are remotable. When placed on a service reference, it indicates
2825 that the interface for the reference is remotable.

2826 The following snippet shows the Java interface for a remotable service with its @Remotable
2827 annotation.

```
2828 package services.hello;
2829
2830 import org.oasisopen.sca.annotation.*;
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
@Remotable
public interface HelloService {

   String hello(String message);
}
```

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

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

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

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

The following snippet shows how a Java service implementation class can use the @Remotable annotation to define a remotable SCA service interface using a Java service interface that is not marked as remotable.

```
package services.hello;

import org.oasisopen.sca.annotation.*;

public interface HelloService {

   String hello(String message);
}

package services.hello;

import org.oasisopen.sca.annotation.*;

@Remotable
@Service(HelloService.class)
public class HelloServiceImpl implements HelloService {

   public String hello(String message) {
         ...
   }
}
```

The following snippet shows how a reference can use the @Remotable annotation to define a remotable SCA service interface using a Java service interface that is not marked as remotable.

```
package services.hello;

import org.oasisopen.sca.annotation.*;

public interface HelloService {
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
2885
2886        String hello(String message);
2887     }
2888
2889     package services.hello;
2890
2891     import org.oasisopen.sca.annotation.*;
2892
2893     public class HelloClient {
2894
2895        @Remotable
2896        @Reference
2897        protected HelloService myHello;
2898
2899        public String greeting(String message) {
2900              return myHello.hello(message);
2901        }
2902     }
2903
```

2904 ## 10.26 @Requires

2905 The following Java code defines the ***@Requires*** annotation:

```
2906
2907     package org.oasisopen.sca.annotation;
2908
2909     import static java.lang.annotation.ElementType.FIELD;
2910     import static java.lang.annotation.ElementType.METHOD;
2911     import static java.lang.annotation.ElementType.PARAMETER;
2912     import static java.lang.annotation.ElementType.TYPE;
2913     import static java.lang.annotation.RetentionPolicy.RUNTIME;
2914
2915     import java.lang.annotation.Inherited;
2916     import java.lang.annotation.Retention;
2917     import java.lang.annotation.Target;
2918
2919     @Inherited
2920     @Retention(RUNTIME)
2921     @Target({TYPE, METHOD, FIELD, PARAMETER})
2922     public @interface Requires {
2923         /**
2924          * Returns the attached intents.
2925          *
2926          * @return the attached intents
2927          */
2928         String[] value() default "";
2929     }
2930
```

2931 The ***@Requires*** annotation supports general purpose intents specified as strings. Users can also
2932 define specific intent annotations using the @Intent annotation.

2933 See the section "General Intent Annotations" for details and samples.

2934 ## 10.27 @Scope

2935 The following Java code defines the ***@Scope*** annotation:

```
2936     package org.oasisopen.sca.annotation;
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Target(TYPE)
@Retention(RUNTIME)
public @interface Scope {

   String value() default "STATELESS";
}
```

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

The @Scope annotation has the following attribute:

- ***value*** – the name of the scope.
  SCA defines the following scope names, but others can be defined by particular Java-based implementation types:
  STATELESS
  COMPOSITE

The default value is STATELESS.

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

```
package services.hello;

import org.oasisopen.sca.annotation.*;

@Service(HelloService.class)
@Scope("COMPOSITE")
public class HelloServiceImpl implements HelloService {

   public String hello(String message) {
         ...
   }
}
```

10.28 @Service

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

```
package org.oasisopen.sca.annotation;

import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Target(TYPE)
@Retention(RUNTIME)
public @interface Service {

   Class<?>[] value();
   String[] names() default {};
}
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

2990 The @Service annotation is used on a component implementation class to specify the SCA services
2991 offered by the implementation. An implementation class need not be declared as implementing all
2992 of the interfaces implied by the services declared in its @Service annotation, but all methods of all
2993 the declared service interfaces MUST be present. [JCA90042] A class used as the implementation
2994 of a service is not required to have a @Service annotation. If a class has no @Service annotation,
2995 then the rules determining which services are offered and what interfaces those services have are
2996 determined by the specific implementation type.

2997 The @Service annotation has the following attributes:

2998
- ***value (1..1)*** – An array of interface or class objects that are exposed as services by this
2999 implementation. If the array is empty, no services are exposed.

3000
- ***names (0..1)*** - An array of Strings which are used as the service names for each of the
3001 interfaces declared in the ***value*** array. The number of Strings in the names attribute array
3002 of the @Service annotation MUST match the number of elements in the value attribute
3003 array. [JCA90050] The value of each element in the @Service names array MUST be
3004 unique amongst all the other element values in the array. [JCA90060]

3005 The ***service name*** of an exposed service defaults to the name of its interface or class, without the
3006 package name. If the names attribute is specified, the service name for each interface or class in
3007 the value attribute array is the String declared in the corresponding position in the names
3008 attribute array.

3009 If a component implementation has two services with the same Java simple name, the names
3010 attribute of the @Service annotation MUST be specified. [JCA90045] If a Java implementation
3011 needs to realize two services with the same Java simple name then this can be achieved through
3012 subclassing of the interface.

3013 The following snippet shows an implementation of the HelloService marked with the @Service
3014 annotation.

```
3015 package services.hello;
3016
3017 import org.oasisopen.sca.annotation.Service;
3018
3019 @Service(HelloService.class)
3020 public class HelloServiceImpl implements HelloService {
3021
3022     public void hello(String name) {
3023         System.out.println("Hello " + name);
3024     }
3025 }
3026
```

Deleted: The array of interfaces or classes specified by the value attribute of the @Service annotation MUST contain at least one element. [JCA90059]

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

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

3027 # 11 WSDL to Java and Java to WSDL

3028 This specification applies the WSDL to Java and Java to WSDL mapping rules as defined by the
3029 JAX-WS 2.1 specification [JAX-WS] for generating remotable Java interfaces from WSDL portTypes
3030 and vice versa.

3031 SCA runtimes MUST support the JAX-WS 2.1 mappings from WSDL to Java and from Java to
3032 WSDL. [JCA100022] For the purposes of the Java-to-WSDL mapping algorithm, the SCA runtime
3033 MUST treat a Java interface as if it had a @WebService annotation on the class, even if it doesn't.
3034 [JCA100001] The SCA runtime MUST treat an @org.oasisopen.sca.annotation.OneWay annotation
3035 as a synonym for the @javax.jws.OneWay annotation. [JCA100002] For the WSDL-to-Java
3036 mapping, the SCA runtime MUST take the generated @WebService annotation to imply that the
3037 Java interface is @Remotable. [JCA100003]

3038 For the mapping from Java types to XML schema types, SCA permits both the JAXB 2.1 [JAX-B]
3039 mapping and the SDO 2.1 [SDO] mapping. SCA runtimes MUST support the JAXB 2.1 mapping
3040 from XML Schema to Java and from Java to XML Schema. [JCA100004] SCA runtimes MAY support
3041 the SDO 2.1 mapping from XML schema types to Java and from Java to XML Schema.
3042 [JCA100005] Having a choice of binding technologies is allowed, as noted in the first paragraph of
3043 section 5 of the JSR 181 (version 2) specification, which is referenced by the JAX-WS specification.

3044 ## 11.1 JAX-WS Annotations and SCA Interfaces

3045 A Java class or interface used to define an SCA interface can contain JAX-WS annotations. In
3046 addition to affecting the Java to WSDL mapping defined by the JAX-WS specification [JAX-WS]
3047 these annotations can impact the SCA interface. An SCA runtime MUST apply the JAX-WS
3048 annotations as described in Table 11-1 and Table 11-2 when introspecting a Java class or interface
3049 class. [JCA100011] This could mean that the interface of a Java implementation is defined by a
3050 WSDL interface declaration.

Annotation	*Property*	*Impact to SCA Interface*
@WebService		A Java interface or class annotated with @WebService MUST be treated as if annotated with the SCA @Remotable annotation [JCA100012]
	name	**If used to define a service, sets service name**
	targetNamespace	None
	serviceName	None
	wsdlLocation	A Java class annotated with the @WebService annotation with its wsdlLocation attribute set MUST have its interface defined by the referenced WSDL definition instead of the annotated Java class. [JCA100013]
	endpointInterface	A Java class annotated with the @WebService annotation with its endpointInterface attribute set MUST have its interface defined by the referenced interface instead of annotated Java class. [JCA100014]
	portName	None
@WebMethod		
	operationName	**Sets operation name**

Formatted: Highlight
Deleted: SCA runtimes MUST support the JAXB 2.1 mapping from Java types to XML schema types.
Deleted: SCA runtimes MAY support the SDO 2.1 mapping from Java types to XML schema types.
Formatted: Bullets and Numbering
Formatted: Highlight
Formatted: Highlight
Formatted: Highlight
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Font: Not Bold, Complex Script Font: Times New Roman, 12 pt
Formatted: Complex Script Font: Times New Roman, 12 pt, Not Bold, Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

	action	None
	exclude	**Method is excluded from the interface.**
@OneWay		The SCA runtime MUST treat an @org.oasisopen.sca.annotation.OneWay annotation as a synonym for the @javax.jws.OneWay annotation. [JCA100002]
@WebParam		
	name	**Sets parameter name**
	targetNamespace	None
	mode	**Sets directionality of parameter**
	header	A Java class or interface containing an @WebParam annotation with its header attribute set to "true" MUST be treated as if the SOAP intent is applied to the Java class or interface. [JCA100015]
	partName	**Overrides name**
@WebResult		
	name	**Sets parameter name**
	targetNamespace	None
	header	A Java class or interface containing an @WebResult annotation with its header attribute set to "true" MUST be treated as if the SOAP intent is applied to the Java class or interface. [JCA100016]
	partName	**Overrides name**
@SOAPBinding		A Java class or interface containing an @SOAPBinding annotation MUST be treated as if the SOAP intent is applied to the Java class or interface. [JCA100021]
	style	
	use	
	parameterStyle	
@HandlerChain		None
	file	
	name	

3051 *Table 11-1: JSR 181 Annotations and SCA Interfaces*

3052

Annotation	*Property*	*Impact to SCA Interface*
@ServiceMode		A Java class containing an @ServiceMode annotation MUST be treated as if the SOAP intent is applied to the Java class. [JCA100017]

Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Complex Script Font: Bold
Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

Annotation	*Property*	*Impact to SCA Interface*
	value	
@WebFault		
	name	**Sets fault name**
	targetNamespace	None
	faultBean	None
@RequestWrapper		None
	localName	
	targetNamespace	
	className	
@ResponseWrapper		None
	localName	
	targetNamespace	
	className	
@WebServiceClient		An interface or class annotated with @WebServiceClient MUST NOT be used to define an SCA interface. [JCA100018]
	name	
	targetNamespace	
	wsdlLocation	
@WebEndpoint		None
	name	
@WebServiceProvider		A class annotated with @WebServiceProvider MUST be treated as if annotated with the SCA @Remotable annotation. [JCA100019]
	wsdlLocation	A Java class annotated with the @WebServiceProvider annotation with its wsdlLocation attribute set MUST have its interface defined by the referenced WSDL definition is used instead of the annotated Java class. [JCA100020]
	serviceName	None
	portName	None
	targetNamespace	None
@BindingType		None
	value	
@WebServiceRef		See JEE specification

Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

Annotation	Property	Impact to SCA Interface
	name	
	wsdlLocation	
	type	
	value	
	mappedName	
@WebServiceRefs		See JEE specification
	value	
@Action		None
	fault	
	input	
	output	
@FaultAction		None
	value	
	output	

3053 Table 11-2: JSR 224 Annotations and SCA Interfaces

3054

Formatted: Indent: Before: 0 pt

Formatted: Bullets and Numbering

3055
11.2 JAX-WS Client Asynchronous API for a Synchronous Service

3056 The JAX-WS specification defines a mapping of a synchronous service invocation, which provides a
3057 client application with a means of invoking that service asynchronously, so that the client can
3058 invoke a service operation and proceed to do other work without waiting for the service operation
3059 to complete its processing. The client application can retrieve the results of the service either
3060 through a polling mechanism or via a callback method which is invoked when the operation
3061 completes.

3062 For SCA service interfaces defined using interface.java, the Java interface MUST NOT contain the
3063 additional client-side asynchronous polling and callback methods defined by JAX-WS. [JCA100006]
3064 For SCA reference interfaces defined using interface.java, the SCA runtime MUST support a Java
3065 interface which contains the additional client-side asynchronous polling and callback methods
3066 defined by JAX-WS. [JCA100007] If the additional client-side asynchronous polling and callback
3067 methods defined by JAX-WS are present in the interface which declares the type of a reference in
3068 the implementation, SCA Runtimes MUST NOT include these methods in the SCA reference
3069 interface in the component type of the implementation. [JCA100008]
3070

3071 The additional client-side asynchronous polling and callback methods defined by JAX-WS are
3072 recognized in a Java interface as follows:

3073 For each method M in the interface, if another method P in the interface has

3074 a. a method name that is M's method name with the characters "Async" appended, and

3075 b. the same parameter signature as M, and

3076 c. a return type of Response<R> where R is the return type of M

3077 then P is a JAX-WS polling method that isn't part of the SCA interface contract.

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

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

a. a method name that is M's method name with the characters "Async" appended, and
b. 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
c. a return type of Future<?>

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

As an example, an interface can be defined in WSDL as follows:

```
<!-- WSDL extract -->
<message name="getPrice">
 <part name="ticker" type="xsd:string"/>
</message>

<message name="getPriceResponse">
 <part name="price" type="xsd:float"/>
</message>

<portType name="StockQuote">
 <operation name="getPrice">
    <input message="tns:getPrice"/>
    <output message="tns:getPriceResponse"/>
 </operation>
</portType>
```

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

```
// asynchronous mapping
@WebService
public interface StockQuote {
 float getPrice(String ticker);
 Response<Float> getPriceAsync(String ticker);
 Future<?> getPriceAsync(String ticker, AsyncHandler<Float>);
}
```

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

```
// synchronous mapping
@WebService
public interface StockQuote {
 float getPrice(String ticker);
}
```

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

11.3 Treatment of SCA Asynchronous Service API

For SCA service interfaces defined using interface.java, the SCA runtime MUST support a Java interface which contains the server-side asynchronous methods defined by SCA. [JCA100010]

Asynchronous service methods are identified as described in the section "Asynchronous handling of Long Running Service Operations" and are mapped to WSDL in the same way as the equivalent synchronous method described in that section.

Formatted: Bullets and Numbering
Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

3127 Generating an asynchronous service method from a WSDL request/response operation follows the
3128 algorithm described in the same section.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

3129 # 12 Conformance

3130 The XML schema pointed to by the RDDL document at the namespace URI, defined by this
3131 specification, are considered to be authoritative and take precedence over the XML schema
3132 defined in the appendix of this document.

3133 Normative code artifacts related to this specification are considered to be authoritative and take
3134 precedence over specification text.

Deleted: For
Deleted: , the specification text is
Deleted: s
Deleted: the code artifacts

3135 There are three categories of artifacts for which this specification defines conformance:

3136 a) SCA Java XML Document,

3137 b) SCA Java Class

3138 c) SCA Runtime.

3139 ## 12.1 SCA Java XML Document

3140 An SCA Java XML document is an SCA Composite Document, or an SCA ComponentType
3141 Document, as defined by the SCA Assembly Model specification [ASSEMBLY], that uses the
3142 <interface.java> element. Such an SCA Java XML document MUST be a conformant SCA
3143 Composite Document or SCA ComponentType Document, as defined by the SCA Assembly Model
3144 specification [ASSEMBLY], and MUST comply with the requirements specified in the Interface
3145 section of this specification.

Deleted: or an SCA ConstrainingType Document
Deleted: or SCA ConstrainingType Document

3146 ## 12.2 SCA Java Class

3147 An SCA Java Class is a Java class or interface that complies with Java Standard Edition version 5.0
3148 and MAY include annotations and APIs defined in this specification. An SCA Java Class that uses
3149 annotations and APIs defined in this specification MUST comply with the requirements specified in
3150 this specification for those annotations and APIs.

3151 ## 12.3 SCA Runtime

3152 The APIs and annotations defined in this specification are meant to be used by Java-based
3153 component implementation models in either partial or complete fashion. A Java-based component
3154 implementation specification that uses this specification specifies which of the APIs and
3155 annotations defined here are used. The APIs and annotations an SCA Runtime has to support
3156 depends on which Java-based component implementation specification the runtime supports. For
3157 example, see the SCA POJO Component Implementation Specification [JAVA_CI].

3158 An implementation that claims to conform to this specification MUST meet the following
3159 conditions:

3160 1. The implementation MUST meet all the conformance requirements defined by the SCA Assembly
3161 Model Specification [ASSEMBLY].

3162 2. The implementation MUST support <interface.java> and MUST comply with all the normative
3163 statements in Section 3.

3164 3. The implementation MUST reject an SCA Java XML Document that does not conform to the sca-
3165 interface-java.xsd schema.

3166 4. The implementation MUST support and comply with all the normative statements in Section 10.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

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

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

   <include schemaLocation="sca-core-1.1-cd04.xsd"/>

   <!-- Java Interface -->
   <element name="interface.java" type="sca:JavaInterface"
            substitutionGroup="sca:interface"/>
   <complexType name="JavaInterface">
      <complexContent>
         <extension base="sca:Interface">
            <sequence>
               <any namespace="##other" processContents="lax" minOccurs="0"
                    maxOccurs="unbounded"/>
            </sequence>
            <attribute name="interface" type="NCName" use="required"/>
            <attribute name="callbackInterface" type="NCName"
                       use="optional"/>
            <attribute name="remotable" type="boolean" use="optional"/>
         </extension>
      </complexContent>
   </complexType>

</schema>
```


Copyright © OASIS® 2005, 2010. All Rights Reserved.

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 SCAClientFactory class 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;

    /**
     * Prevent concrete subclasses from using the no-arg constructor
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
3249         */
3250        private SCAClientFactory() {
3251        }
3252
3253        /**
3254         * Constructor used by concrete subclasses
3255         * @param domainURI - The Domain URI of the Domain accessed via this
3256         * SCAClientFactory
3257         */
3258        protected SCAClientFactory(URI domainURI) {
3259          throws NoSuchDomainException {
3260            this.domainURI = domainURI;
3261        }
3262
3263        /**
3264         * Gets the Domain URI of the Domain accessed via this SCAClientFactory
3265         * @return - the URI for the Domain
3266         */
3267        protected URI getDomainURI() {
3268            return domainURI;
3269        }
3270
3271
3272        /**
3273         * Creates a new instance of the SCAClient that can be
3274         * used to lookup SCA Services.
3275         *
3276         * @param domainURI       URI of the target domain for the SCAClient
3277         * @return A new SCAClient
3278         */
3279        public static SCAClientFactory newInstance( URI domainURI )
3280          throws NoSuchDomainException {
3281            return newInstance(null, null, domainURI);
3282        }
3283
3284        /**
3285         * Creates a new instance of the SCAClient that can be
3286         * used to lookup SCA Services.
3287         *
3288         * @param properties   Properties that may be used when
3289         * creating a new instance of the SCAClient
3290         * @param domainURI       URI of the target domain for the SCAClient
3291         * @return A new SCAClient instance
3292         */
3293        public static SCAClientFactory newInstance(Properties properties,
3294                                                    URI domainURI)
3295          throws NoSuchDomainException {
3296            return newInstance(properties, null, domainURI);
3297        }
3298
3299        /**
3300         * Creates a new instance of the SCAClient that can be
3301         * used to lookup SCA Services.
3302         *
3303         * @param classLoader   ClassLoader that may be used when
3304         * creating a new instance of the SCAClient
3305         * @param domainURI       URI of the target domain for the SCAClient
3306         * @return A new SCAClient instance
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
3307         */
3308        public static SCAClientFactory newInstance(ClassLoader classLoader,
3309                                                   URI domainURI)
3310          throws NoSuchDomainException {
3311            return newInstance(null, classLoader, domainURI);
3312        }
3313
3314        /**
3315         * Creates a new instance of the SCAClient that can be
3316         * used to lookup SCA Services.
3317         *
3318         * @param properties    Properties that may be used when
3319         * creating a new instance of the SCAClient
3320         * @param classLoader   ClassLoader that may be used when
3321         * creating a new instance of the SCAClient
3322         * @param domainURI       URI of the target domain for the SCAClient
3323         * @return A new SCAClient instance
3324         */
3325        public static SCAClientFactory newInstance(Properties properties,
3326                                                   ClassLoader classLoader,
3327                                                   URI domainURI)
3328          throws NoSuchDomainException {
3329            final SCAClientFactoryFinder finder =
3330                factoryFinder != null ? factoryFinder :
3331                      new SCAClientFactoryFinderImpl();
3332            final SCAClientFactory factory
3333                = finder.find(properties, classLoader, domainURI);
3334            return factory;
3335        }
3336
3337        /**
3338         * Returns a reference proxy that implements the business interface <T>
3339         * of a service in the SCA Domain handled by this SCAClientFactory
3340         *
3341         * @param serviceURI the relative URI of the target service. Takes the
3342         * form componentName/serviceName.
3343         * Can also take the extended form componentName/serviceName/bindingName
3344         * to use a specific binding of the target service
3345         *
3346         * @param interfaze The business interface class of the service in the
3347         * domain
3348         * @param <T> The business interface class of the service in the domain
3349         *
3350         * @return a proxy to the target service, in the specified SCA Domain
3351         * that implements the business interface <B>.
3352         * @throws NoServiceException Service requested was not found
3353         * @throws NoSuchDomainException Domain requested was not found
3354         */
3355        public abstract <T> T getService(Class<T> interfaze, String serviceURI)
3356            throws NoSuchServiceException, NoSuchDomainException;
3357    }
```

3358
B.1.2 SCAClientFactoryFinder interface

3359 The SCAClientFactoryFinder interface is a Service Provider Interface representing a SCAClientFactory
3360 finder. SCA provides a default reference implementation of this interface. SCA runtime vendors can
3361 create alternative implementations of this interface that use different class loading or lookup mechanisms.

3362

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
/*
 * 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;

/* 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.
 */
public interface SCAClientFactoryFinder {

      /**
       * Method for finding the SCAClientFactory for a given Domain URI using
       * a specified set of properties and a a specified ClassLoader
       * @param properties - properties to use - may be null
       * @param classLoader - ClassLoader to use - may be null
       * @param domainURI - the Domain URI - must be a valid SCA Domain URI
       * @return - the SCAClientFactory or null if the factory could not be
       * @throws - NoSuchDomainException if the domainURI does not reference
       * a valid SCA Domain
       * found
       */
    SCAClientFactory find(Properties properties,
                          ClassLoader classLoader,
                          URI domainURI )
      throws NoSuchDomainException ;
}
```

B.1.3 SCAClientFactoryFinderImpl class

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

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

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

import org.oasisopen.sca.client.SCAClientFactoryFinder;

import java.io.BufferedReader;
```


Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
3417 import java.io.Closeable;
3418 import java.io.IOException;
3419 import java.io.InputStream;
3420 import java.io.InputStreamReader;
3421 import java.lang.reflect.Constructor;
3422 import java.net.URI;
3423 import java.net.URL;
3424 import java.util.Properties;
3425
3426 import org.oasisopen.sca.NoSuchDomainException;
3427 import org.oasisopen.sca.ServiceRuntimeException;
3428 import org.oasisopen.sca.client.SCAClientFactory;
3429
3430 /**
3431  * This is a default implementation of an SCAClientFactoryFinder which is
3432  * used to find an implementation of the SCAClientFactory interface.
3433  *
3434  * @see SCAClientFactoryFinder
3435  * @see SCAClientFactory
3436  *
3437  * @author OASIS Open
3438  */
3439 public class SCAClientFactoryFinderImpl implements SCAClientFactoryFinder {
3440
3441     /**
3442      * The name of the System Property used to determine the SPI
3443      * implementation to use for the SCAClientFactory.
3444      */
3445     private static final String SCA_CLIENT_FACTORY_PROVIDER_KEY =
3446       SCAClientFactory.class.getName();
3447
3448     /**
3449      * The name of the file loaded from the ClassPath to determine
3450      * the SPI implementation to use for the SCAClientFactory.
3451      */
3452     private static final String SCA_CLIENT_FACTORY_PROVIDER_META_INF_SERVICE
3453         = "META-INF/services/" + SCA_CLIENT_FACTORY_PROVIDER_KEY;
3454
3455     /**
3456      * Public Constructor
3457      */
3458     public SCAClientFactoryFinderImpl() {
3459     }
3460
3461     /**
3462      * Creates an instance of the SCAClientFactorySPI implementation.
3463      * This discovers the SCAClientFactorySPI Implementation and instantiates
3464      * the provider's implementation.
3465      *
3466      * @param properties    Properties that may be used when creating a new
3467      * instance of the SCAClient
3468      * @param classLoader   ClassLoader that may be used when creating a new
3469      * instance of the SCAClient
3470      * @return new instance of the SCAClientFactory
3471      * @throws ServiceRuntimeException Failed to create SCAClientFactory
3472      * Implementation.
3473      */
3474     public SCAClientFactory find(Properties properties,
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
3475                                     ClassLoader classLoader,
3476                                     URI domainURI )
3477         throws NoSuchDomainException, ServiceRuntimeException {
3478               if (classLoader == null) {
3479                     classLoader = getThreadContextClassLoader ();
3480               }
3481               final String factoryImplClassName =
3482                     discoverProviderFactoryImplClass(properties, classLoader);
3483               final Class<? extends SCAClientFactory> factoryImplClass
3484                     = loadProviderFactoryClass(factoryImplClassName,
3485                                                       classLoader);
3486               final SCAClientFactory factory =
3487                     instantiateSCAClientFactoryClass(factoryImplClass,
3488                                                         domainURI );
3489               return factory;
3490         }
3491
3492       /**
3493        * Gets the Context ClassLoader for the current Thread.
3494        *
3495        * @return The Context ClassLoader for the current Thread.
3496        */
3497       private static ClassLoader getThreadContextClassLoader () {
3498           final ClassLoader threadClassLoader =
3499               Thread.currentThread().getContextClassLoader();
3500           return threadClassLoader;
3501       }
3502
3503       /**
3504        * Attempts to discover the class name for the SCAClientFactorySPI
3505        * implementation from the specified Properties, the System Properties
3506        * or the specified ClassLoader.
3507        *
3508        * @return The class name of the SCAClientFactorySPI implementation
3509        * @throw ServiceRuntimeException Failed to find implementation for
3510        * SCAClientFactorySPI.
3511        */
3512       private static String
3513           discoverProviderFactoryImplClass(Properties properties,
3514                                            ClassLoader classLoader)
3515           throws ServiceRuntimeException {
3516           String providerClassName =
3517               checkPropertiesForSPIClassName(properties);
3518           if (providerClassName != null) {
3519               return providerClassName;
3520           }
3521
3522           providerClassName =
3523               checkPropertiesForSPIClassName(System.getProperties());
3524           if (providerClassName != null) {
3525               return providerClassName;
3526           }
3527
3528           providerClassName = checkMETAINFServicesForSIPClassName(classLoader);
3529           if (providerClassName == null) {
3530               throw new ServiceRuntimeException(
3531                     "Failed to find implementation for SCAClientFactory");
3532           }
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
3533
3534            return providerClassName;
3535        }
3536
3537        /**
3538         * Attempts to find the class name for the SCAClientFactorySPI
3539         * implementation from the specified Properties.
3540         *
3541         * @return The class name for the SCAClientFactorySPI implementation
3542         * or <code>null</code> if not found.
3543         */
3544        private static String
3545            checkPropertiesForSPIClassName(Properties properties) {
3546            if (properties == null) {
3547                return null;
3548            }
3549
3550            final String providerClassName =
3551                properties.getProperty(SCA_CLIENT_FACTORY_PROVIDER_KEY);
3552            if (providerClassName != null && providerClassName.length() > 0) {
3553                return providerClassName;
3554            }
3555
3556            return null;
3557        }
3558
3559        /**
3560         * Attempts to find the class name for the SCAClientFactorySPI
3561         * implementation from the META-INF/services directory
3562         *
3563         * @return The class name for the SCAClientFactorySPI implementation or
3564         * <code>null</code> if not found.
3565         */
3566        private static String checkMETAINFServicesForSIPClassName(ClassLoader cl)
3567        {
3568            final URL url =
3569                cl.getResource(SCA_CLIENT_FACTORY_PROVIDER_META_INF_SERVICE);
3570            if (url == null) {
3571                return null;
3572            }
3573
3574            InputStream in = null;
3575            try {
3576                in = url.openStream();
3577                BufferedReader reader = null;
3578                try {
3579                    reader =
3580                      new BufferedReader(new InputStreamReader(in, "UTF-8"));
3581
3582                    String line;
3583                    while ((line = readNextLine(reader)) != null) {
3584                        if (!line.startsWith("#") && line.length() > 0) {
3585                            return line;
3586                        }
3587                    }
3588
3589                    return null;
3590                } finally {
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
3591                    closeStream(reader);
3592                }
3593            } catch (IOException ex) {
3594                throw new ServiceRuntimeException(
3595                            "Failed to discover SCAClientFactory provider", ex);
3596            } finally {
3597                closeStream(in);
3598            }
3599        }
3600
3601        /**
3602         * Reads the next line from the reader and returns the trimmed version
3603         * of that line
3604         *
3605         * @param reader The reader from which to read the next line
3606         * @return The trimmed next line or <code>null</code> if the end of the
3607         * stream has been reached
3608         * @throws IOException I/O error occurred while reading from Reader
3609         */
3610        private static String readNextLine(BufferedReader reader)
3611            throws IOException {
3612
3613            String line = reader.readLine();
3614            if (line != null) {
3615                line = line.trim();
3616            }
3617            return line;
3618        }
3619
3620        /**
3621         * Loads the specified SCAClientFactory Implementation class.
3622         *
3623         * @param factoryImplClassName The name of the SCAClientFactory
3624         * Implementation class to load
3625         * @return The specified SCAClientFactory Implementation class
3626         * @throws ServiceRuntimeException Failed to load the SCAClientFactory
3627         * Implementation class
3628         */
3629        private static Class<? extends SCAClientFactory>
3630            loadProviderFactoryClass(String factoryImplClassName,
3631                                     ClassLoader classLoader)
3632            throws ServiceRuntimeException {
3633
3634            try {
3635                final Class<?> providerClass =
3636                      classLoader.loadClass(factoryImplClassName);
3637                final Class<? extends SCAClientFactory> providerFactoryClass =
3638                    providerClass.asSubclass(SCAClientFactory.class);
3639                return providerFactoryClass;
3640            } catch (ClassNotFoundException ex) {
3641                throw new ServiceRuntimeException(
3642                      "Failed to load SCAClientFactory implementation class "
3643                      + factoryImplClassName, ex);
3644            } catch (ClassCastException ex) {
3645                throw new ServiceRuntimeException(
3646                            "Loaded SCAClientFactory implementation class "
3647                            + factoryImplClassName
3648                        + " is not a subclass of "
```

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```java
                        + SCAClientFactory.class.getName() , ex);
            }
        }

        /**
         * Instantiate an instance of the specified SCAClientFactorySPI
         * Implementation class.
         *
         * @param factoryImplClass The SCAClientFactorySPI Implementation
         * class to instantiate.
         * @return An instance of the SCAClientFactorySPI Implementation class
         * @throws ServiceRuntimeException Failed to instantiate the specified
         * specified SCAClientFactorySPI Implementation class
         */
        private static SCAClientFactory instantiateSCAClientFactoryClass(
                      Class<? extends SCAClientFactory> factoryImplClass,
                      URI domainURI)
            throws NoSuchDomainException, ServiceRuntimeException {

            try {
                Constructor<? extends SCAClientFactory> URIConstructor =
                      factoryImplClass.getConstructor(domainURI.getClass());
                SCAClientFactory provider =
                   URIConstructor.newInstance( domainURI );
                return provider;
            } catch (Throwable ex) {
                throw new ServiceRuntimeException(
                   "Failed to instantiate SCAClientFactory implementation class "
                   + factoryImplClass, ex);
            }
        }

        /**
         * Utility method for closing Closeable Object.
         *
         * @param closeable The Object to close.
         */
        private static void closeStream(Closeable closeable) {
            if (closeable != null) {
                try{
                    closeable.close();
                } catch (IOException ex) {
                    throw new ServiceRuntimeException("Failed to close stream",
                                                      ex);
                }
            }
        }
    }
```

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

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

- Implement their SCAClientFactory implementation class

  Vendors need to provide a subclass of SCAClientFactory that is capable of looking up Services in

Copyright © OASIS® 2005, 2010. All Rights Reserved.

their SCA Runtime. Vendors need to subclass SCAClientFactory and implement the getService() method so that it creates reference proxies to services in SCA Domains handled by their SCA runtime(s).

- Configure the Vendor SCAClientFactory implementation class so that it gets used
  Vendors have several options:

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

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

  Option 2: Provide a META-INF/services file

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

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

  Vendors inject an instance of the vendor implementation of SCAClientFactoryFinder into the factoryFinder field of the SCAClientFactory abstract class. The reference implementation of SCAClientFactoryFinder is not used in this scenario. The vendor implementation of SCAClientFactoryFinder can find the vendor implementation(s) of SCAClientFactory by any means.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

3732 # C. Conformance Items

3733 This section contains a list of conformance items for the SCA-J Common Annotations and APIs
3734 specification.

3735

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]	Where an implementation is used by a "domain level component", and the implementation is marked "Composite" scope, the SCA runtime MUST ensure that all consumers of the component appear to be interacting with a single runtime instance of the implementation.
[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.
[JCA20008]	Where an implementation is marked "Composite" scope and it is used by a component that is nested inside a composite that is used as the implementation of a higher level component, the SCA runtime MUST ensure that all consumers of the component appear to be interacting with a single runtime instance of the implementation. There can be multiple instances of the higher level component, each running on different nodes in a distributed SCA runtime.
[JCA20009]	The SCA runtime MAY use by-reference semantics when passing input parameters, return values or exceptions on calls to remotable services within the same JVM if both the service method implementation and the service proxy used by the client are marked “allows pass by reference”.
[JCA20010]	The SCA runtime MUST use by-value semantics when passing input parameters, return values and exceptions on calls to remotable services within the same JVM if the service method implementation is not marked “allows pass by reference” or the service proxy used by the client is not marked “allows pass by reference”.
[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.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

[JCA30004]	The interface.java element MUST conform to the schema defined in the sca-interface-java.xsd schema.
[JCA30005]	The value of the @remotable attribute on the <interface.java/> element does not override the presence of a @Remotable annotation on the interface class and so if the interface class contains a @Remotable annotation and the @remotable attribute has a value of "false", then the SCA Runtime MUST raise an error and MUST NOT run the component concerned.
[JCA30006]	A Java interface referenced by the @interface attribute of an <interface.java/> element MUST NOT contain any of the following SCA Java annotations: @AllowsPassByReference, @ComponentName, @Constructor, @Context, @Destroy, @EagerInit, @Init, @Intent, @Property, @Qualifier, @Reference, @Scope, @Service.
[JCA30007]	A Java interface referenced by the @callbackInterface attribute of an <interface.java/> element MUST NOT contain any of the following SCA Java annotations: @AllowsPassByReference, @Callback, @ComponentName, @Constructor, @Context, @Destroy, @EagerInit, @Init, @Intent, @Property, @Qualifier, @Reference, @Scope, @Service.
[JCA30009]	The SCA Assembly Model specification [ASSEMBLY] defines a number of criteria that need to be satisfied in order for two interfaces to be compatible or have a compatible superset or subset relationship. If these interfaces are both Java interfaces, compatibility also means that every method that is present in both interfaces is defined consistently in both interfaces with respect to the @OneWay annotation, that is, the annotation is either present in both interfaces or absent in both interfaces.
[JCA30010]	If the identified class is annotated with either the JAX-WS @WebService or @WebServiceProvider annotations and the annotation has a non-empty ***wsdlLocation*** property, then the SCA Runtime MUST act as if an <interface.wsdl/> element is present instead of the <interface.java/> element, with an @interface attribute identifying the portType mapped from the Java interface class and containing @requires and @policySets attribute values equal to the @requires and @policySets attribute values of the <interface.java/> element.
[JCA40001]	The SCA Runtime MUST call a constructor of the component implementation at the start of the Constructing state.
[JCA40002]	The SCA Runtime MUST perform any constructor reference or property injection when it calls the constructor of a component implementation.
[JCA40003]	When the constructor completes successfully, the SCA Runtime MUST transition the component implementation to the Injecting state.
[JCA40004]	If an exception is thrown whilst in the Constructing state, the SCA Runtime MUST transition the component implementation to the Terminated state.
[JCA40005]	When a component implementation instance is in the Injecting state, the SCA Runtime MUST first inject all field and setter properties that are present into the component implementation.
[JCA40006]	When a component implementation instance is in the Injecting state, the SCA Runtime MUST inject all field and setter references that are present into the component implementation, after all the properties have been injected.
[JCA40007]	The SCA Runtime MUST ensure that the correct synchronization model is used so that all injected properties and references are made visible to the component

Deleted: [JCA30008] ... [1]

Formatted: Highlight

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

implementation without requiring the component implementation developer to do any specific synchronization.

[JCA40008]	The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component implementation is in the Injecting state.
[JCA40009]	When the injection of properties and references completes successfully, the SCA Runtime MUST transition the component implementation to the Initializing state.
[JCA40010]	If an exception is thrown whilst injecting properties or references, the SCA Runtime MUST transition the component implementation to the Destroying state.
[JCA40011]	When the component implementation enters the Initializing State, the SCA Runtime MUST call the method annotated with @Init on the component implementation, if present.
[JCA40012]	If a component implementation invokes an operation on an injected reference that refers to a target that has not yet been initialized, the SCA Runtime MUST throw a ServiceUnavailableException.
[JCA40013]	The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component implementation instance is in the Initializing state.
[JCA40014]	Once the method annotated with @Init completes successfully, the SCA Runtime MUST transition the component implementation to the Running state.
[JCA40015]	If an exception is thrown whilst initializing, the SCA Runtime MUST transition the component implementation to the Destroying state.
[JCA40016]	The SCA Runtime MUST invoke Service methods on a component implementation instance when the component implementation is in the Running state and a client invokes operations on a service offered by the component.
[JCA40017]	When the component implementation scope ends, the SCA Runtime MUST transition the component implementation to the Destroying state.
[JCA40018]	When a component implementation enters the Destroying state, the SCA Runtime MUST call the method annotated with @Destroy on the component implementation, if present.
[JCA40019]	If a component implementation invokes an operation on an injected reference that refers to a target that has been destroyed, the SCA Runtime MUST throw an InvalidServiceException.
[JCA40020]	The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component implementation instance is in the Destroying state.
[JCA40021]	Once the method annotated with @Destroy completes successfully, the SCA Runtime MUST transition the component implementation to the Terminated state.
[JCA40022]	If an exception is thrown whilst destroying, the SCA Runtime MUST transition the component implementation to the Terminated state.
[JCA40023]	The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component implementation instance is in the Terminated state.
[JCA40024]	If a property or reference is unable to be injected, the SCA Runtime MUST transition the component implementation to the Destroying state.

Formatted: Highlight
Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

[JCA60001]	When a bidirectional service is invoked, the SCA runtime MUST inject a callback reference for the invoking service into all fields and setter methods of the service implementation class that are marked with a @Callback annotation and typed by the callback interface of the bidirectional service, and the SCA runtime MUST inject null into all other fields and setter methods of the service implementation class that are marked with a @Callback annotation.
[JCA60002]	When a non-bidirectional service is invoked, the SCA runtime MUST inject null into all fields and setter methods of the service implementation class that are marked with a @Callback annotation.
[JCA60003]	The SCA asynchronous service Java interface mapping of a WSDL request-response operation MUST appear as follows: The interface is annotated with the "asyncInvocation" intent. For each service operation in the WSDL, the Java interface contains an operation with - a name which is the JAX-WS mapping of the WSDL operation name, with the suffix "Async" added - a void return type - a set of input parameter(s) which match the JAX-WS mapping of the input parameter(s) of the WSDL operation plus an additional last parameter which is a ResponseDispatch object typed by the JAX-WS Response Bean mapping of the output parameter(s) of the WSDL operation, where ResponseDispatch is the type defined in the SCA Java Common Annotations and APIs specification.
[JCA60004]	An SCA Runtime MUST support the use of the SCA asynchronous service interface for the interface of an SCA service.
[JCA60005]	If the SCA asynchronous service interface ResponseDispatch handleResponse method is invoked more than once through either its sendResponse or its sendFault method, the SCA runtime MUST throw an IllegalStateException.
[JCA60006]	For the purposes of matching interfaces (when wiring between a reference and a service, or when using an implementation class by a component), an interface which has one or more methods which follow the SCA asynchronous service pattern MUST be treated as if those methods are mapped as the equivalent synchronous methods, as follows: Asynchronous service methods are characterized by: f) void return type g) a method name with the suffix "Async" h) a last input parameter with a type of ResponseDispatch<X> i) annotation with the asyncInvocation intent j) possible annotation with the @AsyncFault annotation The mapping of each such method is as if the method had the return type "X", the method name without the suffix "Async" and all the input parameters except the last parameter of the type ResponseDispatch<X>, plus the list of exceptions contained in the @AsyncFault annotation.
[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.
[JCA70002]	Intent annotations MUST NOT be applied to the following: A method of a service implementation class, except for a setter method that is either annotated with @Reference or introspected as an SCA reference according to the

Formatted: Complex Script Font: Arial
Formatted: Highlight
Formatted: Complex Script Font: Arial
Formatted: Highlight
Formatted: Complex Script Font: Arial
Formatted: Highlight
Formatted: Default Paragraph Font, Complex Script Font: 12 pt
Formatted: Highlight
Formatted: Bullets and Numbering
Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

rules in the appropriate Component Implementation specification

A service implementation class field that is not either annotated with @Reference or introspected as an SCA reference according to the rules in the appropriate Component Implementation specification

A service implementation class constructor parameter that is not annotated with @Reference

[JCA70003] Where multiple intent annotations (general or specific) are applied to the same Java element, the SCA runtime MUST compute the combined intents for the Java element by merging the intents from all intent annotations on the Java element according to the SCA Policy Framework [POLICY] rules for merging intents at the same hierarchy level.

[JCA70004] If intent annotations are specified on both an interface method and the method's declaring interface, the SCA runtime MUST compute the effective intents for the method by merging the combined intents from the method with the combined intents for the interface according to the SCA Policy Framework [POLICY] rules for merging intents within a structural hierarchy, with the method at the lower level and the interface at the higher level.

[JCA70005] The @PolicySets annotation MUST NOT be applied to the following:

A method of a service implementation class, except for a setter method that is either annotated with @Reference or introspected as an SCA reference according to the rules in the appropriate Component Implementation specification

A service implementation class field that is not either annotated with @Reference or introspected as an SCA reference according to the rules in the appropriate Component Implementation specification

A service implementation class constructor parameter that is not annotated with @Reference

[JCA70006] If the @PolicySets annotation is specified on both an interface method and the method's declaring interface, the SCA runtime MUST compute the effective policy sets for the method by merging the policy sets from the method with the policy sets from the interface.

[JCA80001] The 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.

[JCA80004] The getServiceReference method MUST throw an IllegalArgumentException if the reference named by the referenceName parameter has multiplicity greater than one.

[JCA80005] The getServiceReference method MUST throw an IllegalArgumentException if the reference named by the referenceName parameter does not have an interface of the type defined by the

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

businessInterface parameter.

[JCA80006]	The getServiceReference method MUST throw an IllegalArgumentException if the component does not have a reference with the name provided in the referenceName parameter.
[JCA80007]	The getServiceReference method MUST return *null* if the multiplicity of the reference named by the referenceName parameter is 0..1 and the reference has no target service configured.
[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 can 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 can 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.
[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 NOT have the value false.
[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.
[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

Deleted: true

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

	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 MUST either continue to work or throw an InvalidServiceException when it is invoked.
[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.
[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.
[JCA90033]	If the target service of a ServiceReference is changed, the reference MUST either continue to work or throw an InvalidServiceException when it is invoked.
[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: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

[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.
[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.
[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.
[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.
[JCA90039]	A reinjected array or Collection for a reference MUST NOT be the same array or Collection object previously injected to the component.
[JCA90040]	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.
[JCA90045]	If a component implementation has two services with the same Java simple name, the names attribute of the @Service annotation MUST be specified.
[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 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.
[JCA90050]	The number of Strings in the names attribute array of the @Service annotation MUST match the number of elements in the value attribute array.
[JCA90052]	The @AllowsPassByReference annotation MUST only annotate the following locations: a service implementation class an individual method of a remotable service implementation an individual reference which uses a remotable interface, where the reference is a field, a setter method, or a constructor parameter
[JCA90053]	The @Remotable annotation is valid only on a Java interface, a Java class, a field, a setter method, or a constructor parameter. It MUST NOT appear anywhere else.
[JCA90054]	When used to annotate a method or a field of an implementation class for injection of a callback object, the type of the method or field MUST be the callback interface

Deleted: A

Deleted: MUST NOT have

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

of at least one bidirectional service offered by the implementation class.

[JCA90055] A method annotated with @OneWay MUST have a void return type and MUST NOT have declared checked exceptions.

[JCA90056] When a method of a Java interface is annotated with @OneWay, the SCA runtime MUST ensure that all invocations of that method are executed in a non-blocking fashion, as described in the section on Asynchronous Programming.

[JCA90057] The @Callback annotation MUST NOT appear on a setter method or a field of a Java implementation class that has COMPOSITE scope.

[JCA90058] When used to annotate a setter method or a field of an implementation class for injection of a callback object, the SCA runtime MUST inject a callback reference proxy into that method or field when the Java class is initialized, if the component is invoked via a service which has a callback interface and where the type of the setter method or field corresponds to the type of the callback interface.

Deleted: [JCA90059] ... [2]

[JCA90060] The value of each element in the @Service names array MUST be unique amongst all the other element values in the array.

[JCA90061] When the Java type of a field, setter method or constructor parameter with the @Property annotation is a primitive type or a JAXB annotated class, the SCA runtime MUST convert a property value specified by an SCA component definition into an instance of the Java type as defined by the XML to Java mapping in the JAXB specification [JAXB] with XML schema validation enabled.

Formatted: Highlight

[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 XML Schema to Java and from Java to XML Schema.

Deleted: from Java types to XML schema types

[JCA100005] SCA runtimes MAY support the SDO 2.1 mapping from XML schema types to Java and from Java to XML Schema.

Deleted: Java types to

[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 SCA runtime MUST support a Java interface which contains the additional client-side asynchronous polling and callback methods defined by JAX-WS.

[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.

[JCA100010] For SCA service interfaces defined using interface.java, the SCA runtime MUST support a Java interface which contains the server-side asynchronous methods defined by SCA.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

[JCA100011]	An SCA runtime MUST apply the JAX-WS annotations as described in Table 11-1 and Table 11-2 when introspecting a Java class or interface class.
[JCA100012]	A Java interface or class annotated with @WebService MUST be treated as if annotated with the SCA @Remotable annotation
[JCA100013]	A Java class annotated with the @WebService annotation with its wsdlLocation attribute set MUST have its interface defined by the referenced WSDL definition instead of the annotated Java class.
[JCA100014]	A Java class annotated with the @WebService annotation with its endpointInterface attribute set MUST have its interface defined by the referenced interface instead of annotated Java class.
[JCA100015]	A Java class or interface containing an @WebParam annotation with its header attribute set to "true" MUST be treated as if the SOAP intent is applied to the Java class or interface.
[JCA100016]	A Java class or interface containing an @WebResult annotation with its header attribute set to "true" MUST be treated as if the SOAP intent is applied to the Java class or interface.
[JCA100017]	A Java class containing an @ServiceMode annotation MUST be treated as if the SOAP intent is applied to the Java class.
[JCA100018]	An interface or class annotated with @WebServiceClient MUST NOT be used to define an SCA interface.
[JCA100019]	A class annotated with @WebServiceProvider MUST be treated as if annotated with the SCA @Remotable annotation.
[JCA100020]	A Java class annotated with the @WebServiceProvider annotation with its wsdlLocation attribute set MUST have its interface defined by the referenced WSDL definition is used instead of the annotated Java class.
[JCA100021]	A Java class or interface containing an @SOAPBinding annotation MUST be treated as if the SOAP intent is applied to the Java class or interface.
[JCA100022]	SCA runtimes MUST support the JAX-WS 2.1 mappings from WSDL to Java and from Java to WSDL.

3736

Formatted: Highlight
Deleted: sdf¶
Formatted: Font color: Black, Highlight
Formatted: Highlight
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Times New Roman, 12 pt
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Complex Script Font: Times New Roman, 12 pt, Not Bold, Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Font: Not Bold, Complex Script Font: Bold
Formatted: Highlight
Formatted: Highlight

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

D. Acknowledgements

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

Participants:

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

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

Scott Vorthmann	TIBCO Software Inc.
Feng Wang	Primeton Technologies, Inc.
Robin Yang	Primeton Technologies, Inc.

3741

3742

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

3743 # E. Non-Normative Text

Copyright © OASIS® 2005, 2010. All Rights Reserved.

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

3744 # F. Revision History

3745 [optional; should not be included in OASIS Standards]

3746

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.

Formatted: Swedish Sweden

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

			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	Issue 104 - RFC2119 work and formal marking of all normative statements - all sections - Completion of Appendix B (list of all normative statements) Accept all changes
cd02-rev4	2009-03-20	Mike Edwards	Editorially removed sentence about componentType side files in Section1 Editorially changed package name to org.oasisopen from org.osoa in lines 291, 292 Issue 6 - add Section 2.3, modify section 9.1 Issue 30 - Section 2.2.2 Issue 76 - Section 6.2.4 Issue 27 - Section 7.6.2, 7.6.2.1 Issue 77 - Section 1.2 Issue 102 - Section 9.21 Issue 123 - conersations removed Issue 65 - Added a new Section 4 ** Causes renumbering of later sections ** ** NB new numbering is used below ** Issue 119 - Added a new section 12 Issue 125 - Section 3.1 Issue 130 - (new number) Section 8.6.2.1 Issue 132 - Section 1 Issue 133 - Section 10.15, Section 10.17 Issue 134 - Section 10.3, Section 10.18 Issue 135 - Section 10.21 Issue 138 - Section 11 Issue 141 - Section 9.1 Issue 142 - Section 10.17.1
cd02-rev5	2009-04-20	Mike Edwards	Issue 154 - Appendix A Issue 129 - Section 8.3.1.1
cd02-rev6	2009-04-28	Mike Edwards	Issue 148 - Section 3 Issue 98 - Section 8
cd02-rev7	2009-04-30	Mike Edwards	Editorial cleanup throughout the spec

Deleted: 1
Deleted: +Issue127c
Deleted: 4
Deleted: Dec
Deleted: 09
Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

cd02-rev8	2009-05-01	Mike Edwards	Further extensive editorial cleanup throughout the spec Issue 160 - Section 8.6.2 & 8.6.2.1 removed
cd02-rev8a	2009-05-03	Simon Nash	Minor editorial cleanup
cd03	2009-05-04	Anish Karmarkar	Updated references and front page clean up
cd03-rev1	2009-09-15	David Booz	Applied Issues: 1,13,125,131,156,157,158,159,161,165,172,177
cd03-rev2	2010-01-19	David Booz	Updated to current Assembly namespace Applied issues: 127,155,168,181,184,185,187,189,190,194

3747

Deleted: 1

Deleted: +Issue127c

Deleted: 4

Deleted: Dec

Deleted: 09

Deleted: 09

Copyright © OASIS® 2005, 2010. All Rights Reserved.

Page 96: [1] Deleted	Dave Booz	1/19/2010 9:06:00 AM

[JCA30008] A Java implementation class referenced by the @interface or the @callbackInterface attribute of an <interface.java/> element MUST NOT contain the following SCA Java annotations:

@Intent, @Qualifier.

Page 103: [2] Deleted	Dave Booz	1/19/2010 9:00:00 AM

[JCA90059] The array of interfaces or classes specified by the value attribute of the @Service annotation MUST contain at least one element.