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

Committee Draft 03 – Rev4

06 February 2010

Deleted: 3

Deleted: 01

Deleted: 5

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: 01

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: 01

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

Notices

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

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

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

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

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

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

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

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

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

Deleted: 01

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: 01

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 34
8.3 Application of Intent Annotations 34
8.3.1 Intent Annotation Examples 35
8.3.2 Inheritance and Annotation 37
8.4 Relationship of Declarative and Annotated Intents 38
8.5 Policy Set Annotations 38
8.6 Security Policy Annotations 39
8.7 Transaction Policy Annotations 40
9 Java API 42
9.1 Component Context 42
9.2 Request Context 47
9.3 ServiceReference Interface 49
9.4 ResponseDispatch interface 50
9.5 ServiceRuntimeException 51
9.6 ServiceUnavailableException 52
9.7 InvalidServiceException 52
9.8 Constants 52
9.9 SCAClientFactory Class 53
9.10 SCAClientFactoryFinder Interface 56
9.11 SCAClientFactoryFinderImpl Class 57
9.12 NoSuchDomainException 58
9.13 NoSuchServiceException 58
10 Java Annotations 59
10.1 @AllowsPassByReference 59
10.2 @AsyncFault 60
10.3 @AsyncInvocation 61
10.4 @Authentication 61
10.5 @Authorization 62
10.6 @Callback 62
10.7 @ComponentName 64
10.8 @Confidentiality 64
10.9 @Constructor 65
10.10 @Context 66
10.11 @Destroy 67
10.12 @EagerInit 67

Deleted: 01

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

10.13 @Init 68
10.14 @Integrity 68
10.15 @Intent 69
10.16 @ManagedSharedTransaction 70
10.17 @ManagedTransaction 70
10.18 @MutualAuthentication 71
10.19 @NoManagedTransaction 72
10.20 @OneWay 72
10.21 @PolicySets 73
10.22 @Property 74
10.23 @Qualifier 75
10.24 @Reference 76
10.24.1 Reinjection 78
10.25 @Remotable 80
10.26 @Requires 82
10.27 @Scope 82
10.28 @Service 83
11 WSDL to Java and Java to WSDL 85
11.1 JAX-WS Annotations and SCA Interfaces 85
11.2 JAX-WS Client Asynchronous API for a Synchronous Service 88
11.3 Treatment of SCA Asynchronous Service API 89
12 Conformance 91
12.1 SCA Java XML Document 91
12.2 SCA Java Class 91
12.3 SCA Runtime 91
A. XML Schema: sca-interface-java.xsd 92
B. Java Classes and Interfaces 93
B.1 SCAClient Classes and Interfaces 93
B.1.1 SCAClientFactory Class 93
B.1.2 SCAClientFactoryFinder interface 95
B.1.3 SCAClientFactoryFinderImpl class 96
B.1.4 SCAClient Classes and Interfaces - what does a vendor need to do? 101
C. Conformance Items 102
D. Acknowledgements 117
E. Revision History 120

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

Deleted: 01

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

Deleted: <#>¶

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/

[JAAS] Java Authentication and Authorization Service Reference Guide

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

Deleted: 01

44 http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.
45 html

46 1.3 Non-Normative References

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

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

Deleted: 01

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]

Snippet 2-1 shows an example of a Java interface for a remotable service:

Deleted: The following snippet

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

Snippet 2-1: Remotable Java Interface

Formatted: Caption

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.

Snippet 2-2 shows an example of a Java interface for a local service:

Deleted: The following snippet

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

Snippet 2-2: Local Java Interface

Formatted: Caption

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: 01

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

2.1.4 @Reference

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

2.1.5 @Property

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

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

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

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

This specification defines two scopes:

- STATELESS
- COMPOSITE

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

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

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

@Destroy specifies a method called when the scope ends.

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

Snippet 2-3 is an example showing a fragment of a service implementation annotated with lifecycle methods:

Deleted: The following snippet

```
@Init
public void start() {
       ...
}

@Destroy
public void stop() {
       ...
}
```

Snippet 2-3: Java Component Implementation with Lifecycle Methods

Formatted: Caption

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

2.2.1 Stateless Scope

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

Deleted: 01

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

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

139 2.2.2 Composite Scope

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

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

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

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

Formatted: Indent: Before: 0 pt, Numbered + Level: 1 + Numbering Style: a, b, c, … + Start at: 1 + Alignment: Left + Aligned at: 36 pt + Tab after: 54 pt + Indent at: 54 pt, Tabs: 18 pt, List tab + Not at 54

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

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

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

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

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

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

168 2.3 @AllowsPassByReference

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

179 The cost of such copying can be very high relative to the cost of making a local call, especially if the data
180 being passed is large. Also, in many cases this copying is not needed if the implementation observes
181 certain conventions for how input parameters, return values and exceptions are used. The
182 @AllowsPassByReference annotation allows service method implementations and client references to be

Deleted: 01

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

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.

2.3.1 Marking Services as “allows pass by reference”

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

- Method execution will not modify any input parameter before the method returns.
- The service implementation will not retain a reference to any mutable input parameter, mutable return value or mutable exception after the method returns.
- The method will observe “allows pass by reference” client semantics (see section 2.3.2) for any callbacks that it makes.

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

2.3.2 Marking References as “allows pass by reference”

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

- The client implementation will not modify any of the method’s input parameters before the method returns. Such modifications might occur in callbacks or separate client threads.
- If the method is one-way, the client implementation will not modify any of the method’s input parameters at any time after calling the method. This is because one-way method calls return immediately without waiting for the service method to complete.

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

2.3.3 Applying “allows pass by reference” to Service Proxies

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

- It is injected into a reference or callback reference that is marked “allows pass by reference”.
- It is obtained by calling ComponentContext.getService() or ComponentContext.getServices() with the name of a reference that is marked “allows pass by reference”.
- It is obtained by calling RequestContext.getCallback() from a service implementation that is marked “allows pass by reference”.
- It is obtained by calling ServiceReference.getService() on a service reference that is marked “allows pass by reference”.

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

- It is injected into a reference or callback reference that is marked “allows pass by reference”.
- It is obtained by calling ComponentContext.getServiceReference() or ComponentContext.getServiceReferences() with the name of a reference that is marked “allows pass by reference”.
- It is obtained by calling RequestContext.getCallbackReference() from a service implementation that is marked “allows pass by reference”.
- It is obtained by calling ComponentContext.cast() on a proxy that is marked “allows pass by reference”.

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

227 ### 2.3.4 Using “allows pass by reference” to Optimize Remotable Calls

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

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

Formatted: Bullets and Numbering

Deleted: 01

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]

Snippet 3-1 is the pseudo-schema for the interface.java element

Deleted: The following

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

Formatted: Caption

Snippet 3-1: interface.java Pseudo-Schema

The interface.java element has the attributes:

Deleted: following

- ***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 class. The value of the @remotable attribute

Deleted: 01

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

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]

Snippet 3-2 shows an example of the Java interface element:

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

Snippet 3-2 Example interface.java Element

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.

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

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

330 3.5 SCA Java Annotations for Interface Classes

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

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

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

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

339 3.6 Compatibility of Java Interfaces

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

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

Deleted: 01

345 # 4 SCA Component Implementation Lifecycle

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

347 ## 4.1 Overview of SCA Component Implementation Lifecycle

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

350 - **The Initialization phase**. This involves constructing an instance of the component implementation
351 class and injecting any properties and references. Once injection is complete, the method annotated
352 with @Init is called, if present, which provides the component implementation an opportunity to
353 perform any internal initialization it requires.
354 - **The Running phase**. This is where the component implementation has been initialized and the SCA
355 Runtime can dispatch service requests to it over its Service interfaces.
356 - **The Destroying phase**. This is where the component implementation's scope has ended and the
357 SCA Runtime destroys the component implementation instance. The SCA Runtime calls the method
358 annotated with @Destroy, if present, which provides the component implementation an opportunity to
359 perform any internal clean up that is required.

360 ## 4.2 SCA Component Implementation Lifecycle State Diagram

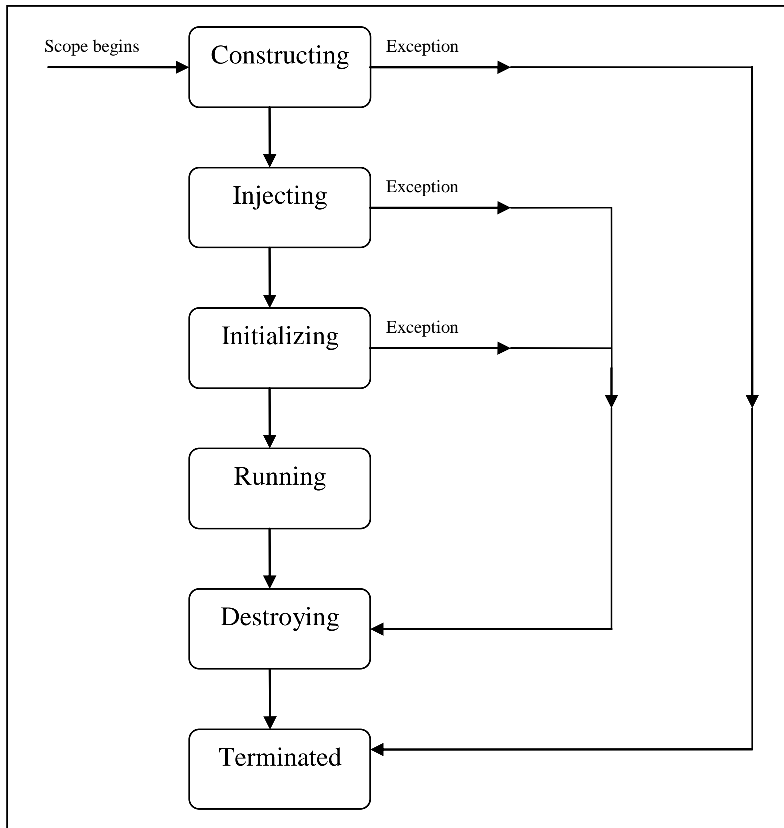
361 The state diagram in Figure 4-1 shows the lifecycle of an SCA component implementation. The sections
362 that follow it describe each of the states that it contains.

Deleted: Figure 4.1

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

Deleted: 01

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



365

366 Figure 4-1: SCA - Component Implementation Lifecycle

Formatted: Caption, Indent: Before: 0 pt, Don't keep with next, Don't keep lines together

Deleted: Figure 4.1 SCA - Component implementation lifecycle¶

367 4.2.1 Constructing State

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

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

373 When the constructor completes successfully, the SCA Runtime MUST transition the component
374 implementation to the Injecting state. [JCA40003] If an exception is thrown whilst in the Constructing
375 state, the SCA Runtime MUST transition the component implementation to the Terminated state.
376 [JCA40004]

377 4.2.2 Injecting State

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

381 When a component implementation instance is in the Injecting state, the SCA Runtime MUST inject all
382 field and setter references that are present into the component implementation, after all the properties
383 have been injected. [JCA40006] The order in which the references are injected is unspecified.

Deleted: 01

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

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

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

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

391 When the injection of properties and references completes successfully, the SCA Runtime MUST
392 transition the component implementation to the Initializing state. [JCA40009] If an exception is thrown
393 whilst injecting properties or references, the SCA Runtime MUST transition the component
394 implementation to the Destroying state. [JCA40010] If a property or reference is unable to be injected, the
395 SCA Runtime MUST transition the component implementation to the Destroying state. [JCA40024]

396 4.2.3 Initializing State

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

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

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

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

409 4.2.4 Running State

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

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

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

417 4.2.5 Destroying State

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

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

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

427 Once the method annotated with @Destroy completes successfully, the SCA Runtime MUST transition
428 the component implementation to the Terminated state. [JCA40021] If an exception is thrown whilst
429 destroying, the SCA Runtime MUST transition the component implementation to the Terminated state.
430 [JCA40022]

Deleted: 01

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

431 ### 4.2.6 Terminated State

432 The lifecycle of the SCA Component has ended.

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

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

Deleted: 01

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.

Snippet 5-1 is a sample of the code that a client would use:

Deleted: The following

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

Deleted: 01

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

```
                                            serviceURI);
                String reply = helloService.sayHello("Mark");

            } catch (Exception e) {
                System.out.println("Received exception");
            }
    }
}
```

Snippet 5-1: Using the SCAClientFactory Interface

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

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

494 # 6 Error Handling

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

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

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

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

Deleted: 01

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.

Snippet 7-1 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.

Deleted: The following example

Deleted: 01

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

```
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();
}
```

Snippet 7-1: Using a Bidirectional Interface

Formatted: Caption

In Snippet 7-1, 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.

Deleted: this example

Snippet 7-2 illustrates a possible implementation of the example service, using the @Callback annotation to request that a callback proxy be injected.

Deleted: The following code snippet

```
@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);
}
```

Snippet 7-2: Example Implementation of a Service with a Bidirectional Interface

Formatted: Caption

Snippet 7-3 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.

Deleted: The code snippet below

```
public class ClientImpl implements ClientService, QuotationCallback {

    private QuotationService myService;

    @Reference
    public void setMyService(QuotationService service) {
        myService = service;
    }
```

Deleted: 01

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

```
    public void aClientMethod() {
        ...
        double quote = myService.requestQuotation("AB123", 2000);
        ...
    }

    public String getState() {
        return "TX";
    }
    public String getZipCode() {
        return "78746";
    }
    public String getCreditRating() {
        return "AA";
    }
}
```

Snippet 7-3: Example Client Using a Biderictional Interface

Snippet 7-3 the callback is ***stateless***, i.e., the callback requests do not need any information relating to the original service request. For a callback that needs information relating to the original service request (a ***stateful*** callback), this information can be passed to the client by the service provider as parameters on the callback request.

Formatted: Caption

Deleted: In this example

7.2.2 Callback Instance Management

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

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

7.2.3 Callback Injection

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. [JCA60001] 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. [JCA60002]

Formatted: Font color: Red

Formatted: Font color: Red

7.2.4 Implementing Multiple Bidirectional Interfaces

Since it is possible for a single implementation class to implement multiple services, it is also possible for callbacks to be defined for each of the services that it implements. The service implementation can include an injected field for each of its callbacks. The runtime injects the callback onto the appropriate field based on the type of the callback. Snippet 7-4 shows the declaration of two fields, each of which corresponds to a particular service offered by the implementation.

Deleted: The following

Deleted: 01

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

```
@Callback
protected MyService1Callback callback1;

@Callback
protected MyService2Callback callback2;
```

Snippet 7-4: Multiple Bidirectional Interfaces in an Implementation

Formatted: Caption

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()`.

Snippet 7-5 comes from a service implementation that uses the callback API:

Deleted: The following example fragments

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

public void someMethod() {

   MyCallback myCallback = callback.getService();    …

   myCallback.receiveResult(theResult);
}
```

Snippet 7-5: Using the Callback API

Formatted: Caption

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. Snippet 7-6 shows how to retrieve a callback in a method programmatically:

Deleted: The snippet below

```
@Context
ComponentContext context;

public void someMethod() {

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

   …

   myCallback.receiveResult(theResult);
}
```

Snippet 7-6: Using RequestContext to get a Callback

Formatted: Caption

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

Deleted: 01

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.

7.4 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 7-7.

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

Snippet 7-7: 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 7-8.

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

Snippet 7-8: 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 7-8.

```
// asynchronous mapping
```

Deleted: snippet 7-1
Deleted: :
Deleted: *Snippet 7-1: Example synchronous Java interface mapping¶*
Formatted: Caption
Deleted: snippet 7-2:
Formatted: Caption
Deleted: Snippet 7-2: Example JAX-WS client asynchronous Java interface mapping¶
Deleted: snippet 7-2:
Deleted: 01

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

750
751
752
753

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

754 *Snippet 7-9: Example SCA Asynchronous Service Java interface Mapping*

755

Formatted: Caption

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

756 The main characteristics of the SCA asynchronous mapping are:

757 • there is a single method, with a name with the string "Async" appended to the operation name

758 • it has a void return type

759 • it has two input parameters, the first is the request message of the operation and the second is a
760 ResponseDispatch object typed by the response message of the operation (following the rules
761 expressed in the JAX-WS specification for the typing of the AsyncHandler object in the client
762 asynchronous API)

763 • it is annotated with the asyncInvocation intent

764 • if the synchronous method has any business faults/exceptions, it is annotated with @AsyncFault,
765 containing a list of the exception classes

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

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

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

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

780 The interface is annotated with the "asyncInvocation" intent.

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

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

784 – a void return type

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

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

792 The ResponseDispatch object passed in as a parameter to a method of a service implementation using
793 the SCA asynchronous service Java interface can be invoked once only through either its sendResponse
794 method or through its sendFault method to return the response resulting from the service method
795 invocation. If the SCA asynchronous service interface ResponseDispatch handleResponse method is
796 invoked more than once through either its sendResponse or its sendFault method, the SCA runtime
797 MUST throw an IllegalStateException. [JCA60005]

798

Deleted: 01

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

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

803 Asynchronous service methods are characterized by:

804 – void return type

805 – a method name with the suffix "Async"

806 – a last input parameter with a type of ResponseDispatch<X>

807 – annotation with the asyncInvocation intent

808 – possible annotation with the @AsyncFault annotation

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

Formatted: Font color: Red

Deleted: 01

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

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 shown in Snippet 8-1.

Deleted: as follows:

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

Snippet 8-1: Intent Format

Formatted: Caption

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 those in Snippet 8-2.

Deleted: the following

Deleted: :

Deleted: 01

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

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

Snippet 8-2: Example Intent Constants

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) is shown in Snippet 8-3:

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

Snippet 8-3: Multiple Intnets in One Annotation

The annotation in Snippet 8-3 attaches the intents "confidentiality.message" and "integrity.message".

Snippet 8-4 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) {
      …
   }
}
```

Snippet 8-4: Annotation a Reference

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, the definition of Snippet 8-4 would instead look like Snippet 8-5.

```
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) {
```

Formatted: Caption
Deleted: follows
Formatted: Caption
Deleted: This
Deleted: The following
Formatted: Caption
Deleted: this
Deleted: this
Deleted: :
Deleted: 01

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

```
        …
    }
}
```

Snippet 8-5: Using Intent Constants and strings

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

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

where

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

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

8.2 Specific Intent Annotations

In addition to the general intent annotation supplied by the @Requires annotation described in section 8.2, it is also possible to have Java annotations that correspond to specific policy intents. SCA provides a number of these specific intent annotations and it is also possible to create new specific intent annotations for any intent.

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

For example, the SCA confidentiality intent described in the section on General Intent Annotations using the @Requires(CONFIDENTIALITY) annotation can also be specified with the @Confidentiality specific intent annotation. The specific intent annotation for the "integrity" security intent is shown in Snippet 8-6.

```
@Integrity
```

Snippet 8-6: Example Specific Intent Annotation

An example of a qualified specific intent for the "authentication" intent is shown in Snippet 8-7.

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

Snippet 8-7: Example Qualified Specific Intent Annotation

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

The general form of specific intent annotations is shown in Snippet 8-8

```
'@' Intent ('(' qualifiers ')')?
```

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

```
Intent          ::= NCName
qualifiers      ::= '"' qualifier '"' ('," ' qualifier '"')*
qualifier ::= NCName ('.' qualifier)?
```

Snippet 8-8: Specific Intent Annotation Format

Formatted: Caption
Deleted: above
Deleted: :
Formatted: Caption
Deleted: :
Formatted: Caption
Deleted: :
Formatted: Caption
Deleted: 01

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

8.2.1 How to Create Specific Intent Annotations

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

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

Alternatively, the QName of the intent can be specified using separate parameters for the targetNamespace and the localPart, as shown in Snippet 8-9:

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

Snippet 8-9: Defining a Specific Intent Annotation

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

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

In this case, the attribute's definition needs to be marked with the @Qualifier annotation. The @Qualifier tells SCA that the value of the attribute is treated as a qualifier for the intent represented by the whole annotation. If more than one qualifier value is specified in an annotation, it means that multiple qualified forms exist. For example the annotation in Snippet 8-10

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

Snippet 8-10: Multiple Qualifiers in an Annotation'

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

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

[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 is shown in Snippet 8-11:

Deleted: follows

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

Snippet 8-11: Multiple Policy Annotations

Formatted: Caption

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.

Snippet 8-12 shows how intents on references are merged. In this example, the intents for `myRef` are "authentication" and "confidentiality.message".

Deleted: Example 8.1

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

Snippet 8-12: Merging Intents on References

Formatted: Caption

Deleted: Example 8.1. Merging intents on references.¶

Snippet 8-13 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".

Deleted: Example 8.2

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

Snippet 8-13: Mutually Exclusive Intents

Formatted: Caption

Deleted: Example 8.2. Mutually exclusive intents.¶

Snippet 8-14 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".

Deleted: Example 8.3

```
@Authentication
```


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

Deleted: 01

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

Snippet 8-14: Intents on Java Interfaces, Interface Methods, and Java Classes

Snippet 8-15 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() {...}
}
```

Snippet 8-15: Intents on Java Service Implementation Classes

Snippet 8-16 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;
}
```

Snippet 8-16: Intents on Java References and their Interfaces and Methods

Snippet 8-17 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() {...}
```


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

```
}
```

Snippet 8-17: Intent on Implementation Method

Snippet 8-18 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();
}
```

Snippet 8-18: Merging Qualified and Unqualified Intents on Java Interfaces and Methods

Snippet 8-19 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();
}
```

Snippet 8-19: Merging Mutually Exclusive Intents on Java Interfaces and Methods

8.3.2 Inheritance and Annotation

Snippet 8-20 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() {...}
}

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) {...}
```


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

```
    @Authentication
    String helloWorld() {...}
}
```

Snippet 8-20: Usage example of Annotated Policy and Inheritance

Formatted: Caption

Deleted: 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 shows the equivalent declarative security interaction policy of the methods of the HelloService and HelloChildService implementations corresponding to the Java classes shown in Snippet 8-20.

Deleted: Table 8.1 below

Deleted: 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 Snippet 8-20

Deleted: ¶ *Table 8.1.*

Formatted: Keep with next, Don't adjust space between Latin and Asian text

8.4 Relationship of Declarative and Annotated Intents

Deleted: Declarative intents equivalent to annotated intents in Example 8.9.¶

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.

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:

Deleted:

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

Snippet 8-21: PolicySet Annotation Format

Formatted: Caption

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

Deleted: 01

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

1186 An example of the @PolicySets annotation is shown in Snippet 8-22:

1187

```
1188        @Reference(name="helloService", required=true)
1189        @PolicySets({ MY_NS + "WS_Encryption_Policy",
1190                      MY_NS + "WS_Authentication_Policy" })
1191        public setHelloService(HelloService service) {
1192           . . .
1193        }
```

1194 *Snippet 8-22: Use of @PolicySets*

Formatted: Caption

1195

1196 In this case, the Policy Sets WS_Encryption_Policy and WS_Authentication_Policy are applied, both
1197 using the namespace defined for the constant MY_NS.

1198 PolicySets need to satisfy intents expressed for the implementation when both are present, according to
1199 the rules defined in the Policy Framework specification [POLICY].

1200 The SCA Policy Set annotation can be applied to the following Java elements:

1201 • Java class

1202 • Java interface

1203 • Method

1204 • Field

1205 • Constructor parameter

1206 The @PolicySets annotation MUST NOT be applied to the following:

1207 • A method of a service implementation class, except for a setter method that is either annotated with
1208 @Reference or introspected as an SCA reference according to the rules in the appropriate
1209 Component Implementation specification

1210 • A service implementation class field that is not either annotated with @Reference or introspected as
1211 an SCA reference according to the rules in the appropriate Component Implementation specification

1212 • A service implementation class constructor parameter that is not annotated with @Reference

1213 [JCA70005]

1214 The @PolicySets annotation can be applied to classes, interfaces, and interface methods. Applying a
1215 @PolicySets annotation to a field, setter method, or constructor parameter allows policy sets to be
1216 defined at references. The @PolicySets annotation can also be applied to reference interfaces and their
1217 methods.

1218 If the @PolicySets annotation is specified on both an interface method and the method's declaring
1219 interface, the SCA runtime MUST compute the effective policy sets for the method by merging the policy
1220 sets from the method with the policy sets from the interface. [JCA70006] This merging process does not
1221 remove or change any policy sets that are applied to the interface.

1222 8.6 Security Policy Annotations

1223 This section introduces annotations for commonly used SCA security intents, as defined in the SCA
1224 Policy Framework Specification [POLICY]. Also see the SCA Policy Framework Specification for
1225 additional security policy intents that can be used with the @Requires annotation. The following
1226 annotations for security policy intents and qualifiers are defined:

1227 • @Authentication

1228 • @Authorization

1229 • @Confidentiality

1230 • @Integrity

1231 • @MutualAuthentication

Deleted: 01

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

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".

Snippet 8-23 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!");
         …
   }
}
```

Snippet 8-23: Usage of Security Intents on a Reference

8.7 Transaction Policy Annotations

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

The @ManagedTransaction intent has the following Qualifiers:

- global

Deleted: The following example
Deleted:
Formatted: Caption
Deleted: ¶ Example 8.10. Usage of security intents on a reference.¶ ¶
Deleted: ¶
Deleted: 01

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

1286 • local

1287 The formal definitions of the transaction intent annotations are found in the section "Java Annotations".

1288 Snippet 8-24 shows an example of applying a transaction intent to a component implementation, where
1289 the component implementation requires a global transaction.

1290

```
1291 package services.hello;
1292 // Interface for HelloService
1293 public interface HelloService {
1294    String hello(String helloMsg);
1295 }
1296
1297 // Implementation class for HelloService
1298 package services.hello.impl;
1299
1300 import services.hello.HelloService;
1301 import org.oasisopen.sca.annotation.*;
1302
1303 @Service(HelloService.class)
1304 @ManagedTransaction("global")
1305 public class HelloServiceImpl implements HelloService {
1306
1307    public void someMethod() {
1308          …
1309    }
1310 }
```

1311 *Snippet 8-24: Usage of Transaction Intents in an Implementation*

1312

Deleted: ¶

Deleted: The following example

Formatted: Caption

Deleted: ¶ Example 8.11. Usage of transaction intents in an implementation.¶ ¶

Deleted: 01

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

Figure 9-1 defines the ***ComponentContext*** interface:

Deleted: The following Java code

```
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;

}
```

Figure 9-1: ComponentContext Interface

Deleted: *9-1*

getURI () method:

Returns the absolute URI of the component within the SCA Domain.

Returns:

- ***String*** which contains the absolute URI of the component in the SCA Domain
  The ComponentContext.getURI method MUST return the absolute URI of the component in the SCA Domain. [JCA80008]

Formatted: Pattern: Clear (Yellow)

Parameters:

- ***none***

Exceptions:

- ***none***

getService (Class<B> businessInterface, String referenceName) method:

Deleted: 01

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

Returns a typed service proxy object for a reference defined by the current component, where the reference has multiplicity 0..1 or 1..1.

Returns:

- ***B*** which is a proxy object for the reference, which implements the interface B contained in the businessInterface parameter.

  The ComponentContext.getService method MUST return the proxy object implementing the interface provided by the businessInterface parameter, for the reference named by the referenceName parameter with the interface defined by the businessInterface parameter when that reference has a target service configured. [JCA80009]

  The ComponentContext.getService 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. [JCA80010]

Parameters:

- ***Class<B> businessInterface*** - the Java interface for the service reference
- ***String referenceName*** - the name of the service reference

Exceptions:

- 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]
- The ComponentContext.getService method MUST throw an IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter. [JCA80011]
- The ComponentContext.getService method MUST throw an IllegalArgumentException if the service reference with the name supplied in the referenceName does not have an interface compatible with the interface supplied in the businessInterface parameter. [JCA80012]

getServiceReference (Class<B> businessInterface, String referenceName) method:

Returns a ServiceReference object for a reference defined by the current component, where the reference has multiplicity 0..1 or 1..1.

Returns:

- ***ServiceReference<B>*** which is a ServiceReference proxy object for the reference, which implements the interface contained in the businessInterface parameter.
  The ComponentContext.getServiceReference method MUST return a ServiceReference object typed by the interface provided by the businessInterface parameter, for the reference named by the referenceName parameter with the interface defined by the businessInterface parameter when that reference has a target service configured. [JCA80013]
  The ComponentContext.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]

Parameters:

- ***Class<B> businessInterface*** - the Java interface for the service reference
- ***String referenceName*** - the name of the service reference

Exceptions:

- The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if the reference named by the referenceName parameter has multiplicity greater than one. [JCA80004]
- The ComponentContext.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]

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

- The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if the component does not have a reference with the name provided in the referenceName parameter. [JCA80006]

getServices(Class<B> businessInterface, String referenceName) method:

Returns a list of typed service proxies for a reference defined by the current component, where the reference has multiplicity 0..n or 1..n.

Returns:

- ***Collection<B>*** which is a collection of proxy objects for the reference, one for each target service to which the reference is wired, where each proxy object implements the interface B contained in the businessInterface parameter.

  The ComponentContext.getServices method MUST return a collection containing one proxy object implementing the interface provided by the businessInterface parameter for each of the target services configured on the reference identified by the referenceName parameter. [JCA80014]

  The ComponentContext.getServices method MUST return an empty collection if the service reference with the name supplied in the referenceName parameter is not wired to any target services. [JCA80015]

Parameters:

- ***Class<B> businessInterface*** - the Java interface for the service reference
- ***String referenceName*** - the name of the service reference

Exceptions:

- The ComponentContext.getServices method MUST throw an IllegalArgumentException if the reference identified by the referenceName parameter has multiplicity of 0..1 or 1..1. [JCA80016]
- The ComponentContext.getServices method MUST throw an IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter. [JCA80017]
- The ComponentContext.getServices method MUST throw an IllegalArgumentException if the service reference with the name supplied in the referenceName does not have an interface compatible with the interface supplied in the businessInterface parameter.[JCA80018]

getServiceReferences(Class<B> businessInterface, String referenceName) method:

Returns a list of typed ServiceReference objects for a reference defined by the current component, where the reference has multiplicity 0..n or 1..n.

Returns:

- ***Collection<ServiceReference<B>>*** which is a collection of ServiceReference objects for the reference, one for each target service to which the reference is wired, where each proxy object implements the interface B contained in the businessInterface parameter. The collection is empty if the reference is not wired to any target services.

  The ComponentContext.getServiceReferences method MUST return a collection containing one ServiceReference object typed by the interface provided by the businessInterface parameter for each of the target services configured on the reference identified by the referenceName parameter. [JCA80019]

  The ComponentContext.getServiceReferences method MUST return an empty collection if the service reference with the name supplied in the referenceName parameter is not wired to any target services. [JCA80020]

Parameters:

- ***Class<B> businessInterface*** - the Java interface for the service reference

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

1455 • ***String referenceName*** - the name of the service reference

1456 Exceptions:

1457 • The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if
1458 the reference identified by the referenceName parameter has multiplicity of 0..1 or 1..1. [JCA80021]

1459 • The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if
1460 the component does not have a reference with the name supplied in the referenceName parameter.
1461 [JCA80022]

1462 • The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if
1463 the service reference with the name supplied in the referenceName does not have an interface
1464 compatible with the interface supplied in the businessInterface parameter. [JCA80023]

1465

1466 ***createSelfReference(Class<B> businessInterface) method:***

1467 Returns a ServiceReference object that can be used to invoke this component over the designated
1468 service.

1469 Returns:

1470 • ***ServiceReference<B>*** which is a ServiceReference object for the service of this component which
1471 has the supplied business interface. If the component has multiple services with the same business
1472 interface the SCA runtime can return a ServiceReference for any one of them.

1473 The ComponentContext.createSelfReference method MUST return a ServiceReference object typed
1474 by the interface defined by the businessInterface parameter for one of the services of the invoking
1475 component which has the interface defined by the businessInterface parameter. [JCA80024]

1476 Parameters:

1477 • ***Class<B> businessInterface*** - the Java interface for the service

1478 Exceptions:

1479 • The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if
1480 the component does not have a service which implements the interface identified by the
1481 businessInterface parameter. [JCA80025]

1482

1483 ***createSelfReference(Class<B> businessInterface, String serviceName) method:***

1484 Returns a ServiceReference that can be used to invoke this component over the designated service. The
1485 serviceName parameter explicitly declares the service name to invoke

1486 Returns:

1487 • ***ServiceReference<B>*** which is a ServiceReference proxy object for the reference, which implements
1488 the interface contained in the businessInterface parameter.

1489 The ComponentContext.createSelfReference method MUST return a ServiceReference object typed
1490 by the interface defined by the businessInterface parameter for the service identified by the
1491 serviceName of the invoking component and which has the interface defined by the businessInterface
1492 parameter. [JCA80026]

1493 Parameters:

1494 • ***Class<B> businessInterface*** - the Java interface for the service reference

1495 • ***String serviceName*** - the name of the service reference

1496 Exceptions:

1497 • The ComponentContext.createSelfReference method MUST throw an IllegalArgumentException if the
1498 component does not have a service with the name identified by the serviceName parameter.
1499 [JCA80027]

Formatted: Pattern: Clear (Yellow)

Deleted: 01

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

- The ComponentContext.createSelfReference method MUST throw an IllegalArgumentException if the component service with the name identified by the serviceName parameter does not implement a business interface which is compatible with the supplied businessInterface parameter. [JCA80028]

getProperty (Class<B> type, String propertyName) method:

Returns the value of an SCA property defined by this component.

Returns:

- ***<B>*** which is an object of the type identified by the type parameter containing the value specified for the property in the SCA configuration of the component. ***null*** if the SCA configuration of the component does not specify any value for the property.

  The ComponentContext.getProperty method MUST return an object of the type identified by the type parameter containing the value specified in the component configuration for the property named by the propertyName parameter or null if no value is specified in the configuration. [JCA80029]

Parameters:

- ***Class<B> type*** - the Java class of the property (Object mapped type for primitive Java types - e.g. Integer if the type is int)
- ***String propertyName*** - the name of the property

Exceptions:

- The ComponentContext.getProperty method MUST throw an IllegalArgumentException if the component does not have a property with the name identified by the propertyName parameter. [JCA80030]
- The ComponentContext.getProperty method MUST throw an IllegalArgumentException if the component property with the name identified by the propertyName parameter does not have a type which is compatible with the supplied type parameter. [JCA80031]

getRequestContext() method:

Returns the RequestContext for the current SCA service request.

Returns:

- ***RequestContext*** which is the RequestContext object for the current SCA service invocation. ***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]

Parameters:

- ***none***

Exceptions:

- ***none***

cast(B target) method:

Casts a type-safe reference to a ServiceReference

Returns:

- ***ServiceReference<B>*** which is a ServiceReference object which implements the same business interface B as a reference proxy object

  The ComponentContext.cast method MUST return a ServiceReference object which is typed by the same business interface as specified by the reference proxy object supplied in the target parameter. [JCA80032]

Formatted: Pattern: Clear (Yellow)

Formatted: Pattern: Clear (Yellow)

Deleted: The ComponentContext.getProperty method MUST return an object of the type identified by the type parameter containing the value specified in the component configuration for the property named by the propertyName parameter or null if no value is specified in the configuration.

Formatted: Pattern: Clear (Yellow)

Deleted: The ComponentContext.getProperty method MUST throw an IllegalArgumentException if the component does not have a property with the name identified by the propertyName parameter.

Formatted: Pattern: Clear (Yellow)

Deleted: The ComponentContext.getProperty method MUST throw an IllegalArgumentException if the component property with the name identified by the propertyName parameter does not have a type which is compatible with the supplied type parameter.

Formatted: Pattern: Clear (Yellow)

Deleted: The ComponentContext.getRequestContext method MUST return a non-null RequestContext object 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.

Deleted: <#>¶

Formatted: Pattern: Clear (Yellow)

Deleted: 01

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

Parameters:

- ***B target*** - a type safe reference proxy object which implements the business interface B

Exceptions:

- The ComponentContext.cast method MUST return a ServiceReference object which is typed by the same business interface as specified by the reference proxy object supplied in the target parameter. [JCA80033]

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(..).***

Snippet 9-1 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");
}
```

Snippet 9-1: ComponentContext Injection Example

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.

9.2 Request Context

Figure 9-2 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();
    }
```

Figure 9-2: RequestContext Interface

getSecuritySubject ( ) method:

Returns the JAAS Subject of the current request (see the JAAS Reference Guide [JAAS] for details of JAAS).

Returns:

- ***javax.security.auth.Subject*** object which is the JAAS subject for the request.

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

null if there is no subject for the request.

The RequestContext.getSecuritySubject method MUST return the JAAS subject of the current request, or null if there is no subject or null if the method is invoked from code not processing a service request or callback request. [JCA80034]

Formatted: Font: Bold, Italic

Formatted: Pattern: Clear (Yellow)

Parameters:

- ***none***

Exceptions:

- ***none***

getServiceName ( ) method:

Returns the name of the service on the Java implementation the request came in on.

Returns:

- ***String*** containing the name of the service. ***null*** if the method is invoked from a thread that is not processing a service operation or a callback operation.

  The RequestContext.getServiceName method MUST return the name of the service for which an operation is being processed, or null if invoked from a thread that is not processing a service operation or a callback operation. [JCA80035]

Formatted: Bullets and Numbering

Formatted: Pattern: Clear (Yellow)

Parameters:

- ***none***

Exceptions:

- ***none***

getCallbackReference ( ) method:

Returns a service reference proxy for the callback for the invoked service operation, as specified by the service client.

Returns:

- ***ServiceReference<CB>*** which is a service reference for the callback for the invoked service, as supplied by the service client. It is typed with the callback interface.

  null if the invoked service has an interface which is not bidirectional or if the getCallbackReference() method is called during the processing of a callback operation.

  null if the method is invoked from a thread that is not processing a service operation.

  The RequestContext.getCallbackReference method MUST return a ServiceReference object typed by the interface of the callback supplied by the client of the invoked service, or null if either the invoked service is not bidirectional or if the method is invoked from a thread that is not processing a service operation. [JCA80036]

Formatted: Pattern: Clear (Yellow)

Parameters:

- ***none***

Exceptions:

- ***none***

getCallback ( ) method:

Returns a proxy for the callback for the invoked service as specified by the service client.

Returns:

Deleted: 01

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

- ***CB*** proxy object for the callback for the invoked service as supplied by the service client. It is typed with the callback interface.

  null if the invoked service has an interface which is not bidirectional or if the getCallback() method is called during the processing of a callback operation.

  null if the method is invoked from a thread that is not processing a service operation.

  The RequestContext.getCallback method MUST return a reference proxy object typed by the interface of the callback supplied by the client of the invoked service, or null if either the invoked service is not bidirectional or if the method is invoked from a thread that is not processing a service operation. [JCA80037]

Parameters:

- ***none***

Exceptions:

- ***none***

getServiceReference ( ) method:

Returns a ServiceReference object for the service that was invoked.

Returns:

- ***ServiceReference<B>*** which is a service reference for the invoked service. It is typed with the interface of the service.

  null if the method is invoked from a thread that is not processing a service operation or a callback operation.

  When invoked during the execution of a service operation, the RequestContext.getServiceReference method MUST return a ServiceReference that represents the service that was invoked. [JCA80003]

  When invoked during the execution of a callback operation, the RequestContext.getServiceReference method MUST return a ServiceReference that represents the callback that was invoked. [JCA80038]

  When invoked from a thread not involved in the execution of either a service operation or of a callback operation, the RequestContext.getServiceReference method MUST return null. [JCA80039]

Parameters:

- ***none***

Exceptions:

- ***none***

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.

9.3 ServiceReference Interface

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.

Figure 9-3 defines the ***ServiceReference*** interface:

```
package org.oasisopen.sca;

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


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

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

Figure 9-3: ServiceReference Interface

getService ( ) method:

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.

Returns:

- ***<B>*** which is type-safe reference proxy object to the target of this reference. It is typed with the interface of the target service.

  The ServiceReference.getService method MUST return a reference proxy object which can be used to invoke operations on the target service of the reference and which is typed with the business interface of the reference. [JCA80040]

Parameters:

- ***none***

Exceptions:

- ***none***

getBusinessInterface ( ) method:

Returns the Java class for the business interface associated with this ServiceReference.

Returns:

- ***Class<B>*** which is a Class object of the business interface associated with the reference.

  The ServiceReference.getBusinessInterface method MUST return a Class object representing the business interface of the reference. [JCA80041]

Parameters:

- ***none***

Exceptions:

- ***none***

9.4 ResponseDispatch interface

The ***ResponseDispatch*** interface is shown in Figure 9-4:

```
package org.oasisopen.sca;

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

Figure 9-4: ResponseDispatch Interface

sendResponse (T response) method:

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

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***

  The ResponseDispatch.sendResponse() method MUST send the response message to the client of an asynchronous service. [JCA50057]

Parameters:

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

Exceptions:

- The ResponseDispatch.sendResponse() method MUST throw an InvalidStateException if either the sendResponse method or the sendFault method has already been called once. [JCA80058]

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***

  The ResponseDispatch.sendFault() method MUST send the supplied fault to the client of an asynchronous service. [JCA80059]

Parameters:

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

Exceptions:

- The ResponseDispatch.sendFault() method MUST throw an InvalidStateException if either the sendResponse method or the sendFault method has already been called once. [JCA80060]

getContext () method:

Obtains the context object for the ResponseDispatch method

Returns:

- ***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

Figure 9-5 shows the ***ServiceRuntimeException***.

```
package org.oasisopen.sca;

public class ServiceRuntimeException extends RuntimeException {
   …
```


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

```
}
```

Figure 9-5: ServiceRuntimeException

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

9.6 ServiceUnavailableException

Figure 9-6 shows the ***ServiceUnavailableException***.

```
package org.oasisopen.sca;

public class ServiceUnavailableException extends ServiceRuntimeException {
   …
}
```

Figure 9-6: ServiceUnavailableException

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

Figure 9-7 shows the ***InvalidServiceException***.

```
package org.oasisopen.sca;

public class InvalidServiceException extends ServiceRuntimeException {
   …
}
```

Figure 9-7: InvalidServiceException

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.

9.8 Constants

The SCA ***Constants*** interface defines a number of constant values that are used in the SCA Java APIs and Annotations. Figure 9-8 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+"}";
}
```

Figure 9-8: Constants Interface

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

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 shown in Figure 9-9:

```
/*
 * 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)
   throws NoSuchDomainException {...}

    public URI getDomainURI() {...}

    public static SCAClientFactory newInstance( URI domainURI )
         throws NoSuchDomainException {...}

    public static SCAClientFactory newInstance(Properties properties,
                                                             URI domainURI)
         throws NoSuchDomainException {...}

    public static SCAClientFactory newInstance(ClassLoader classLoader,
                                                             URI domainURI)
         throws NoSuchDomainException {...}

    public static SCAClientFactory newInstance(Properties properties,
                                         ClassLoader classLoader,
                                         URI domainURI)
```

Deleted: *Figure 9-9: Constants interface*¶
¶

Deleted: Figure 9-10

Deleted: 01

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

```
        throws NoSuchDomainException {...}

    public abstract <T> T getService(Class<T> interfaze, String serviceURI)
        throws NoSuchServiceException, NoSuchDomainException;
}
```

Figure 9-9: SCAClientFactory Class

newInstance (URI domainURI) method:

Obtains a object implementing the SCAClientFactory class.

Returns:

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

  The SCAClientFactory.newInstance(URI) method MUST return an object which implements the SCAClientFactory class for the SCA Domain identified by the domainURI parameter. [JCA80042]

Parameters:

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

Exceptions:

- The SCAClientFactory.newInstance(URI) method MUST throw a NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain. [JCA80043]

newInstance(Properties properties, URI domainURI) method:

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

Returns:

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

  The SCAClientFactory.newInstance(Properties, URI) method MUST return an object which implements the SCAClientFactory class for the SCA Domain identified by the domainURI parameter. [JCA80044]

Parameters:

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

Exceptions:

- The SCAClientFactory.newInstance(Properties, URI) method MUST throw a NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain. [JCA80045]

newInstance(Classloader classLoader, URI domainURI) method:

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

Returns:

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

  The SCAClientFactory.newInstance(Classloader, URI) method MUST return an object which implements the SCAClientFactory class for the SCA Domain identified by the domainURI parameter. [JCA80046]

Parameters:

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

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

1917 • ***domainURI*** - a URI for the SCA Domain which is targeted by the returned SCAClient object

1918 Exceptions:

1919 • The SCAClientFactory.newInstance(Classloader, URI) method MUST throw a
1920 NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.
1921 [JCA80047]

Formatted: Pattern: Clear (Yellow)

1922

1923 ***newInstance(Properties properties, Classloader classLoader, URI domainURI) method:***

1924 Obtains a object implementing the SCAClientFactory class using a specified set of properties and a
1925 specified classloader.

1926 Returns:

1927 • ***object*** which implements the SCAClientFactory class

1928 The SCAClientFactory.newInstance(Properties, Classloader, URI) method MUST return an object
1929 which implements the SCAClientFactory class for the SCA Domain identified by the domainURI
1930 parameter. [JCA80048]

Formatted: Pattern: Clear (Yellow)

1931 Parameters:

1932 • ***properties*** - a set of Properties that can be used when creating the object which implements the
1933 SCAClientFactory class.

1934 • ***classLoader*** - a ClassLoader to use when creating the object which implements the
1935 SCAClientFactory class.

1936 • ***domainURI*** - a URI for the SCA Domain which is targeted by the returned SCAClient object

1937 Exceptions:

1938 • The SCAClientFactory.newInstance(Properties, Classloader, URI) MUST throw a
1939 NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.
1940 [JCA80049]

Formatted: Pattern: Clear (Yellow)

1941

1942 ***getService(Class<T> interfaze, String serviceURI) method:***

1943 Obtains a proxy reference object for a specified target service in a specified SCA Domain.

1944 Returns:

1945 • ***<T>*** a proxy object which implements the business interface T
1946 Invocations of a business method of the proxy causes the invocation of the corresponding operation
1947 of the target service.

1948 The SCAClientFactory.getService method MUST return a proxy object which implements the
1949 business interface defined by the interfaze parameter and which can be used to invoke operations on
1950 the service identified by the serviceURI parameter. [JCA80050]

Formatted: Pattern: Clear (Yellow)

1951 Parameters:

1952 • ***interfaze*** - a Java interface class which is the business interface of the target service

1953 • ***serviceURI*** - a String containing the relative URI of the target service within its SCA Domain.

1954 Takes the form componentName/serviceName or can also take the extended form
1955 componentName/serviceName/bindingName to use a specific binding of the target service

1956 Exceptions:

1957 • The SCAClientFactory.getService method MUST throw a NoSuchServiceException if a service with
1958 the relative URI serviceURI and a business interface which matches interfaze cannot be found in the
1959 SCA Domain targeted by the SCAClient object. [JCA80051]

Formatted: Pattern: Clear (Yellow)

1960 • The SCAClientFactory.getService method MUST throw a NoSuchServiceException if the domainURI
1961 of the SCAClientFactory does not identify a valid SCA Domain. [JCA80052]

Formatted: Pattern: Clear (Yellow)

1962

Deleted: 01

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

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

getDomainURI() method:

Obtains the Domain URI value for this SCAClientFactory

Returns:

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

  The SCAClientFactory.getDomainURI method MUST return the SCA Domain URI of the Domain associated with the SCAClientFactory object. [JCA80053]

Parameters:

- ***none***

Exceptions:

- The SCAClientFactory.getDomainURI method MUST throw a ***NoSuchServiceException*** if the domainURI of the SCAClientFactory does not identify a valid SCA Domain. [JCA80054]

private SCAClientFactory() method:

This private no-argument constructor prevents instantiation of an SCAClientFactory instance without the use of the constructor with an argument, even by subclasses of the abstract SCAClientFactory class.

factoryFinder protected field:

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.

Deleted: ¶

9.10 SCAClientFactoryFinder Interface

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

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

public interface SCAClientFactoryFinder {

    SCAClientFactory find(Properties properties,
                          ClassLoader classLoader,
                          URI domainURI )
        throws NoSuchDomainException ;
}
```

Figure 9-10: SCAClientFactoryFinder Interface

Formatted: Caption

Deleted: Figure 9-11: SCAClientFactoryFinder interface

Formatted: Normal, Indent: First line: 0 pt

find (Properties properties, ClassLoader classloader, URI domainURI) method:

Obtains an implementation of the SCAClientFactory interface.

Returns:

- ***SCAClientFactory*** implementation object

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

The implementation of the SCAClientFactoryFinder.find method MUST return an object which is an implementation of the SCAClientFactory interface, for the SCA Domain represented by the doaminURI parameter, using the supplied properties and classloader. [JCA80055]

Parameters:

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

Exceptions:

- The implementation of the SCAClientFactoryFinder.find method MUST throw a ServiceRuntimeException if the SCAClientFactory implementation could not be found. [JCA80056]

9.11 SCAClientFactoryFinderImpl Class

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

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

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

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

Snippet 9-2: SCAClientFactoryFinderImpl Class

SCAClientFactoryFinderImpl () method:

Public constructor for the SCAClientFactoryFinderImpl.

Returns:

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

Parameters:

- ***none***

Exceptions:

- ***none***

find (Properties, ClassLoader, URI) method:

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

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

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

Returns:

- ***SCAClientFactory*** implementation object

Parameters:

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

Exceptions:

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

9.12 NoSuchDomainException

Figure 9-11 shows the ***NoSuchDomainException***:

```
package org.oasisopen.sca;

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

Figure 9-11: NoSuchDomainException Class

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

9.13 NoSuchServiceException

Figure 9-12 shows the ***NoSuchServiceException***:

```
package org.oasisopen.sca;

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

Figure 9-12: NoSuchServiceException Class

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

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]

Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)

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]

Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)

10.1 @AllowsPassByReference

Figure 10-1 defines the ***@AllowsPassByReference*** annotation:

Deleted: The following Java code

```
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;
}
```

Figure 10-1: AllowsPassByReference Annotation

Formatted: Caption

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 attribute:

Deleted: following

- ***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:

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

Deleted: 01

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.
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".

Snippet 10-1 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) {
   …
}
```

Snippet 10-1: Use of @AllowsPassByReference on a Method

Snippet 10-2 shows a sample where @AllowsPassByReference is defined for a client reference of a Java component implementation class.

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

Snippet 10-2: Use of @AllowsPassByReference on a Reference

10.2 @AsyncFault

Figure 10-2 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 {};

}
```

Figure 10-2: AsyncFault Annotation

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

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

10.3 @AsyncInvocation

Figure 10-3 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;
}
```

Figure 10-3: AsyncInvocation Annotation

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";
```

Deleted: ¶
Deleted: The following Java code
Formatted: Caption
Deleted: ¶
Deleted: 01

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

```
    String AUTHENTICATION_MESSAGE = AUTHENTICATION + ".message";
    String AUTHENTICATION_TRANSPORT = AUTHENTICATION + ".transport";

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

Figure 10-4: Authentication Annotation

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

Figure 10-5 defines the @Authorization annotation:

Formatted: Caption

Deleted: The following Java code

```
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";
}
```

Figure 10-5: Authorization Annotation

Formatted: Caption

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

Figure 10-6 defines the ***@Callback*** annotation:

Deleted: The following Java code

Deleted: 01

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

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

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

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

Figure 10-6: Callback Annotation

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 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]

Snippet 10-3 shows an example use of the @Callback annotation to declare a callback interface.

```
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);
}
```

Snippet 10-3: Use of @Callback

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

2342 The implied component type is for Snippet 10-3 is shown in Snippet 10-4.

2343

```
2344 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" >
2345
2346    <service name="MyService">
2347       <interface.java interface="somepackage.MyService"
2348                 callbackInterface="somepackage.MyServiceCallback"/>
2349    </service>
2350 </componentType>
```

2351 *Snippet 10-4: Implied componentType for Snippet 10-3*

2352 10.7 @ComponentName

2353 Figure 10-7 defines the ***@ComponentName*** annotation:

2354

```
2355 package org.oasisopen.sca.annotation;
2356
2357 import static java.lang.annotation.ElementType.FIELD;
2358 import static java.lang.annotation.ElementType.METHOD;
2359 import static java.lang.annotation.ElementType.TYPE;
2360 import java.lang.annotation.Retention;
2361 import java.lang.annotation.Target;
2362
2363 @Target({METHOD, FIELD})
2364 @Retention(RUNTIME)
2365 public @interface ComponentName {
2366
2367 }
```

2368 *Figure 10-7: ComponentName Annotation*

2369

2370 The @ComponentName annotation is used to denote a Java class field or setter method that is used to
2371 inject the component name.

2372 Snippet 10-5 shows a component name field definition sample.

2373

```
2374 @ComponentName
2375 private String componentName;
```

2376 *Snippet 10-5: Use of @ComponentName on a Field*

2377

2378 Snippet 10-6 shows a component name setter method sample.

2379

```
2380 @ComponentName
2381 public void setComponentName(String name) {
2382  //…
2383 }
```

2384 *Snippet 10-6: Use of @ComponentName on a Setter*

2385 10.8 @Confidentiality

2386 Figure 10-8 defines the ***@Confidentiality*** annotation:

2387

Deleted: In this example, t
Deleted: :
Formatted: Caption, Don't adjust space between Latin and Asian text
Deleted: The following Java code
Formatted: Caption
Deleted: The following snippet
Formatted: Caption
Deleted: The following snippet
Formatted: Caption
Deleted: The following Java code
Deleted: 01

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

```
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 {
    String CONFIDENTIALITY = SCA_PREFIX + "confidentiality";
    String CONFIDENTIALITY_MESSAGE = CONFIDENTIALITY + ".message";
    String CONFIDENTIALITY_TRANSPORT = CONFIDENTIALITY + ".transport";

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

Figure 10-8: Confidentiality Annotation

The ***@Confidentiality*** annotation is used to indicate the need for confidentiality. 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.

10.9 @Constructor

Figure 10-9 defines the ***@Constructor*** annotation:

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

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

@Target(CONSTRUCTOR)
@Retention(RUNTIME)
public @interface Constructor { }
```

Figure 10-9: Constructor Annotation

The @Constructor annotation is used to mark a particular constructor to use when instantiating a Java component implementation. If a constructor of an implementation class is annotated with @Constructor

Formatted: Caption
Deleted: The following Java code
Formatted: Caption
Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)
Deleted: 01

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

and the constructor has parameters, each of these parameters MUST have either a @Property annotation or a @Reference annotation. [JCA90003]

Snippet 10-7 shows a sample for the @Constructor annotation.

Deleted: The following snippet

```
public class HelloServiceImpl implements HelloService {

   public HelloServiceImpl(){
    ...
    }

    @Constructor
    public HelloServiceImpl(@Property(name="someProperty")
                                String someProperty ){
    ...
    }

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

Formatted: Caption

Snippet 10-7: Use of @Constructor

10.10 @Context

Figure 10-10 defines the ***@Context*** annotation:

Deleted: The following Java code

```
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 {

}
```

Formatted: Caption

Figure 10-10: Context Annotation

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.

Snippet 10-8 shows a ComponentContext field definition sample.

Deleted: The following snippet

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

Formatted: Caption

Snippet 10-8: Use of @Context for a ComponentContext

Deleted: 01

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

2491 Snippet 10-9 shows a RequestContext field definition sample.

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

Snippet 10-9: Use of @Context for a RequestContext

10.11 @Destroy

Figure 10-11 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 {

}
```

Figure 10-11: Destroy Annotation

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

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. [JCA90005]

Snippet 10-10 shows a sample for a destroy method definition.

```
@Destroy
public void myDestroyMethod() {
  …
}
```

Snippet 10-10: Use of @Destroy

10.12 @EagerInit

Figure 10-12: EagerInit Annotation defines the ***@EagerInit*** 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)
```

Deleted: The following snippet
Formatted: Caption
Deleted: The following Java code
Formatted: Caption
Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)
Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)
Deleted: The following snippet
Formatted: Caption
Deleted: The following Java code
Deleted: 01

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

```
public @interface EagerInit {

}
```

Figure 10-12: EagerInit Annotation

Formatted: Caption

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

Formatted: Complex Script Font: Times New Roman, 9 pt, Pattern: Clear (Yellow)

10.13 @Init

Figure 10-13: Init Annotation defines the **@Init** annotation:

Deleted: The following Java code

```
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 Init {


}
```

Figure 10-13: Init Annotation

Formatted: Caption

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

Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)

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. [JCA90009]

Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)

Snippet 10-11 shows an example of an init method definition.

Deleted: The following snippet

```
@Init
public void myInitMethod() {
  …
}
```

Snippet 10-11: Use of @Init

Formatted: Caption

10.14 @Integrity

Figure 10-14 defines the **@Integrity** annotation:

Deleted: The following Java code

```
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;
```

Deleted: 01

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

```
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(Integrity.INTEGRITY)
public @interface Integrity {
    String INTEGRITY = SCA_PREFIX + "integrity";
    String INTEGRITY_MESSAGE = INTEGRITY + ".message";
    String INTEGRITY_TRANSPORT = INTEGRITY + ".transport";

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

Figure 10-14: Integrity Annotation

Formatted: Caption

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

10.15 @Intent

Figure 10-15 defines the **@Intent** annotation:

Deleted: The following Java code

```
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 "";

    /**
```

Deleted: 01

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

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

Figure 10-15: Intent Annotation

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

Figure 10-16 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
@Target({TYPE, FIELD, METHOD, PARAMETER})
@Retention(RUNTIME)
@Intent(ManagedSharedTransaction.MANAGEDSHAREDTRANSACTION)
public @interface ManagedSharedTransaction {
    String MANAGEDSHAREDTRANSACTION = SCA_PREFIX + "managedSharedTransaction";
}
```

Figure 10-16: ManagedSharedTransaction Annotation

The ***@ManagedSharedTransaction*** annotation is used to indicate the need for a distributed and globally coordinated ACID transaction. 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.

10.17 @ManagedTransaction

Figure 10-17 defines the @ManagedTransaction annotation:

```
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
```


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

2692
2693
2694
2695
2696
2697
2698
2699
2700
2701

```
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
```

2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720

```
 * 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 "";
}
```

Formatted: Caption

2721 *Figure 10-17: ManagedTransaction Annotation*

2722

2723 The ***@ManagedTransaction*** annotation is used to indicate the need for an ACID transaction. See the
2724 SCA Policy Framework Specification [POLICY] for details on the meaning of the intent. See the section
2725 on Application of Intent Annotations for samples of how intent annotations are used in Java.

2726
10.18 @MutualAuthentication

2727 Figure 10-18 defines the @MutualAuthentication annotation:

Deleted: The following Java code

2728

2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748

```
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 @MutualAuthentication annotation is used to indicate that
 * a mutual authentication policy is needed.
 */
@Inherited
@Target({TYPE, FIELD, METHOD, PARAMETER})
@Retention(RUNTIME)
```

Deleted: 01

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

```
@Intent(MutualAuthentication.MUTUALAUTHENTICATION)
public @interface MutualAuthentication {
    String MUTUALAUTHENTICATION = SCA_PREFIX + "mutualAuthentication";
}
```

Figure 10-18: MutualAuthentication Annotation

The ***@MutualAuthentication*** annotation is used to indicate the need for mutual authentication between a service consumer and a service provider. 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.

10.19 @NoManagedTransaction

Figure 10-19 defines the @NoManagedTransaction 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 @NoManagedTransaction annotation is used to indicate that
 * a non-transactional environment is needed.
 */
@Inherited
@Target({TYPE, FIELD, METHOD, PARAMETER})
@Retention(RUNTIME)
@Intent(NoManagedTransaction.NOMANAGEDTRANSACTION)
public @interface NoManagedTransaction {
    String NOMANAGEDTRANSACTION = SCA_PREFIX + "noManagedTransaction";
}
```

Figure 10-19: NoManagedTransaction Annotation

The ***@NoManagedTransaction*** annotation is used to indicate that the component does not want to run in an ACID transaction. 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.

10.20 @OneWay

Figure 10-20 defines the ***@OneWay*** annotation:

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

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

Formatted: Caption
Deleted: The following Java code
Formatted: Caption
Deleted: The following Java code
Deleted: 01

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

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

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


}
```

Formatted: Caption

Figure 10-20: OneWay Annotation

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

Formatted: Font color: Red

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. [JCA90056]

Formatted: Font color: Red

The @OneWay annotation has no attributes.

Snippet 10-12 shows the use of the @OneWay annotation on an interface.

Deleted: The following snippet

```
package services.hello;

import org.oasisopen.sca.annotation.OneWay;

public interface HelloService {
    @OneWay
    void hello(String name);
}
```

Formatted: Caption

Snippet 10-12: Use of @OneWay

10.21 @PolicySets

Figure 10-21 defines the ***@PolicySets*** annotation:

Deleted: The following Java code

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

@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 "";
}
```

Formatted: Caption

Figure 10-21: PolicySets Annotation

Deleted: 01

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

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

Figure 10-22 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;
}
```

Figure 10-22: Property Annotation

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 attributes:

- ***name (0..1)*** – 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 @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 (0..1)*** – 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]

}

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

Snippet 10-13 shows a property field definition sample.

Deleted: The following snippet

```
@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 ) {
    ....
}
```

Snippet 10-13: Use of @Property on a Field

Formatted: Caption

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]

Snippet 10-14 shows the definition of a configuration property using the @Property annotation for a collection.

Deleted: The following snippet

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

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

Snippet 10-14: Use of @Property with a Collection

Formatted: Caption

10.23 @Qualifier

Figure 10-23 defines the ***@Qualifier*** annotation:

Deleted: The following Java code

```
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 {
}
```

Figure 10-23: Qualifier Annotation

Formatted: Caption

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]

Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)

Deleted: 01

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

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

10.24 @Reference

Figure 10-24 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;
}
```

Figure 10-24: Reference Annotation

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 attributes:

- ***name : String (0..1)*** – 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 (0..1)*** – 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]

Snippet 10-15 shows a reference field definition sample.

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

Snippet 10-15: Use of @Reference on a Field

Snippet 10-16 shows a reference setter sample

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


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

```
    ...
}
```

Snippet 10-16: Use of @Reference on a Setter

Snippet 10-17 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!");
         …
}
```

Snippet 10-17: Use of @Reference and a ServiceReference

The presence of a @Reference annotation is reflected in the componentType information that the runtime generates through reflection on the implementation class. Snippet 10-18 shows the component type for the component implementation fragment in Snippet 10-17.

```
<?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>
```

Snippet 10-18: Implied componentType for Implementation in Snippet 10-17

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]

Snippet 10-19 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)
```


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

```
    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!");
           }
           …
    }
```

Snippet 10-19: Use of @Reference with a List of ServiceReferences

Snippet 10-20 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>
```

Snippet 10-20: Implied componentType for Implementation in Snippet 10-19

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]

If an operation is called on a reference where the target of that reference has been undeployed, the SCA runtime SHOULD throw an InvalidServiceException. [JCA90027] 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. [JCA90028] If the target service of the reference is changed, the reference MUST either continue to work or throw an InvalidServiceException when it is invoked. [JCA90029] If it doesn't work, the exception thrown will depend on the runtime and the cause of the failure.

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

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. [JCA90030] If the target of a ServiceReference has been undeployed, the SCA runtime SHOULD throw a InvalidServiceException when an operation is invoked on the ServiceReference. [JCA90031] If the target of a ServiceReference has become unavailable, the SCA runtime SHOULD throw a ServiceUnavailableException when an operation is invoked on the ServiceReference. [JCA90032] If the target service of a ServiceReference is changed, the reference MUST either continue to work or throw an InvalidServiceException when it is invoked. [JCA90033] If it doesn't work, the exception thrown will depend on the runtime and the cause of the failure.

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. [JCA90034] 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. [JCA90035] 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. [JCA90036]

The rules for reference reinjection also apply to references with a multiplicity of 0..n or 1..n. This means that 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. [JCA90037] 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. [JCA90038] A reinjected array or Collection for a reference MUST NOT be the same array or Collection object previously injected to the component. [JCA90039]

	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 becomes unavailable	Business methods throw ServiceUnavailableException	Business methods throw ServiceUnavailableException	Result is be a reference to the unavailable service. Business methods throw ServiceUnavailableException.
Target service changed	might continue to work, depending on the runtime and the type of change that was made. If it doesn't work, the	might continue to work, depending on the runtime and the type of change that was made. If it doesn't work, the	Result is a reference to the changed service.

Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)
Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)
Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)
Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)
Formatted: Complex Script Font: 9 pt, Pattern: Clear (Yellow)
Formatted: Pattern: Clear (Yellow)
Formatted: Pattern: Clear (Yellow)
Deleted: 01

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

	exception thrown will depend on the runtime and the cause of the failure.	exception thrown will depend on the runtime and the cause of the failure.	
* 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().			

Table 10-1Reinjection Effects

Formatted: Caption

10.25 @Remotable

Figure 10-25 defines the ***@Remotable*** annotation:

Deleted: The following Java code

```
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 Remotable {

}
```

Figure 10-25: Remotable Annotation

Formatted: Caption

The @Remotable annotation is used to indicate that an SCA service interface is remotable. 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. [JCA90053] A remotable service can be published externally as a service and MUST be translatable into a WSDL portType. [JCA90040]

Formatted: Font color: Red

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

Snippet 10-21 shows the Java interface for a remotable service with its @Remotable annotation.

Deleted: The following snippet

```
package services.hello;

import org.oasisopen.sca.annotation.*;

@Remotable
public interface HelloService {

   String hello(String message);
}
```


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

Deleted: 01

3169 Snippet 10-21: Use of @Remotable on an Interface

3170

3171 The style of remotable interfaces is typically ***coarse grained*** and intended for ***loosely coupled***
3172 interactions. Remotable service interfaces are not allowed to make use of method ***overloading***.

3173 Complex data types exchanged via remotable service interfaces need to be compatible with the
3174 marshalling technology used by the service binding. For example, if the service is going to be exposed
3175 using the standard Web Service binding, then the parameters can be JAXB [JAX-B] types or they can be
3176 Service Data Objects (SDOs) [SDO].

3177 Independent of whether the remotable service is called from outside of the composite that contains it or
3178 from another component in the same composite, the data exchange semantics are ***by-value***.

3179 Implementations of remotable services can modify input data during or after an invocation and can modify
3180 return data after the invocation. If a remotable service is called locally or remotely, the SCA container is
3181 responsible for making sure that no modification of input data or post-invocation modifications to return
3182 data are seen by the caller.

3183 Snippet 10-22 shows how a Java service implementation class can use the @Remotable annotation to
3184 define a remotable SCA service interface using a Java service interface that is not marked as remotable.

3185

```
3186 package services.hello;
3187
3188 import org.oasisopen.sca.annotation.*;
3189
3190 public interface HelloService {
3191
3192    String hello(String message);
3193 }
3194
3195 package services.hello;
3196
3197 import org.oasisopen.sca.annotation.*;
3198
3199 @Remotable
3200 @Service(HelloService.class)
3201 public class HelloServiceImpl implements HelloService {
3202
3203    public String hello(String message) {
3204          ...
3205    }
3206 }
```

3207 Snippet 10-22: Use of @Remotable on a Class

3208

3209 Snippet 10-23 shows how a reference can use the @Remotable annotation to define a remotable SCA
3210 service interface using a Java service interface that is not marked as remotable.

3211

```
3212 package services.hello;
3213
3214 import org.oasisopen.sca.annotation.*;
3215
3216 public interface HelloService {
3217
3218    String hello(String message);
3219 }
3220
3221 package services.hello;
3222
3223 import org.oasisopen.sca.annotation.*;
```

Formatted: Caption

Deleted: The following snippet

Formatted: Caption

Deleted: The following snippet

Deleted: 01

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

```
public class HelloClient {

   @Remotable
   @Reference
   protected HelloService myHello;

   public String greeting(String message) {
          return myHello.hello(message);
   }
}
```

Snippet 10-23: Use of @Remotable on a Reference

10.26 @Requires

Figure 10-26 defines the ***@Requires*** 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 java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Inherited
@Retention(RUNTIME)
@Target({TYPE, METHOD, FIELD, PARAMETER})
public @interface Requires {
    /**
     * Returns the attached intents.
     *
     * @return the attached intents
     */
    String[] value() default "";
}
```

Figure 10-26: Requires Annotation

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

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

10.27 @Scope

Figure 10-27 defines the ***@Scope*** 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;
```


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

```
@Target(TYPE)
@Retention(RUNTIME)
public @interface Scope {

   String value() default "STATELESS";
}
```

Figure 10-27: Scope Annotation

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 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.

Snippet 10-24 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) {
         ...
   }
}
```

Snippet 10-24: Use of @Scope

10.28 @Service

Figure 10-28 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 {};
   }
```

Formatted: Caption

Deleted: following

Deleted: The following snippet

Formatted: Caption

Deleted: The following Java code

Deleted: 01

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

3326 *Figure 10-28: Service Annotation*

3327

3328 The @Service annotation is used on a component implementation class to specify the SCA services
3329 offered by the implementation. An implementation class need not be declared as implementing all of the
3330 interfaces implied by the services declared in its @Service annotation, but all methods of all the declared
3331 service interfaces MUST be present. [JCA90042] A class used as the implementation of a service is not
3332 required to have a @Service annotation. If a class has no @Service annotation, then the rules
3333 determining which services are offered and what interfaces those services have are determined by the
3334 specific implementation type.

3335 The @Service annotation has the attributes:

3336 • ***value (1..1)*** – An array of interface or class objects that are exposed as services by this
3337 implementation. If the array is empty, no services are exposed.

3338 • ***names (0..1)*** - An array of Strings which are used as the service names for each of the interfaces
3339 declared in the ***value*** array. The number of Strings in the names attribute array of the @Service
3340 annotation MUST match the number of elements in the value attribute array. [JCA90050] The value of
3341 each element in the @Service names array MUST be unique amongst all the other element values in
3342 the array. [JCA90060]

3343 The ***service name*** of an exposed service defaults to the name of its interface or class, without the
3344 package name. If the names attribute is specified, the service name for each interface or class in the
3345 value attribute array is the String declared in the corresponding position in the names attribute array.

3346 If a component implementation has two services with the same Java simple name, the names attribute of
3347 the @Service annotation MUST be specified. [JCA90045] If a Java implementation needs to realize two
3348 services with the same Java simple name then this can be achieved through subclassing of the interface.

3349 Snippet 10-25 shows an implementation of the HelloService marked with the @Service annotation.

3350

```
3351 package services.hello;
3352
3353 import org.oasisopen.sca.annotation.Service;
3354
3355 @Service(HelloService.class)
3356 public class HelloServiceImpl implements HelloService {
3357
3358     public void hello(String name) {
3359         System.out.println("Hello " + name);
3360     }
3361 }
```

3362 *Snippet 10-25: Use of @Service*

Formatted: Caption
Deleted: following
Formatted: Font color: Auto
Formatted: Font color: Auto
Deleted: The following snippet
Formatted: Caption
Deleted: 01

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

11 WSDL to Java and Java to WSDL

This specification applies the WSDL to Java and Java to WSDL mapping rules as defined by the JAX-WS 2.1 specification [JAX-WS] for generating remotable Java interfaces from WSDL portTypes and vice versa.

SCA runtimes MUST support the JAX-WS 2.1 mappings from WSDL to Java and from Java to WSDL. [JCA100022] For the purposes of the Java-to-WSDL mapping algorithm, the SCA runtime MUST treat a Java interface as if it had a @WebService annotation on the class, even if it doesn't. [JCA100001] The SCA runtime MUST treat an @org.oasisopen.sca.annotation.OneWay annotation as a synonym for the @javax.jws.OneWay annotation. [JCA100002] For the WSDL-to-Java mapping, the SCA runtime MUST take the generated @WebService annotation to imply that the Java interface is @Remotable. [JCA100003]

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

11.1 JAX-WS Annotations and SCA Interfaces

A Java class or interface used to define an SCA interface can contain JAX-WS annotations. In addition to affecting the Java to WSDL mapping defined by the JAX-WS specification [JAX-WS] these annotations can impact the SCA interface. 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. [JCA100011] This could mean that the interface of a Java implementation is defined by a 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: Font color: Red

Formatted: Font color: Red

Formatted: Font color: Red

Formatted: Font color: Red

Formatted: Font color: Red

Deleted: 01

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	

Formatted: Font color: Red

Formatted: Font color: Red

Formatted: Font color: Red

3386 *Table 11-1: JSR 181 Annotations and SCA Interfaces*

Deleted: *11-1*

3387

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: Font color: Red

Deleted: 01

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: Font color: Red

Formatted: Font color: Red

Formatted: Font color: Red

Deleted: 01

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	

Table 11-2: JSR 224 Annotations and SCA Interfaces

Deleted: *11-2*

Deleted: ¶

11.2 JAX-WS Client Asynchronous API for a Synchronous Service

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

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

The additional client-side asynchronous polling and callback methods defined by JAX-WS are recognized in a Java interface according to the steps:

Deleted: ¶

Deleted: as follows

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

a. a method name that is M's method name with the characters "Async" appended, and
b. the same parameter signature as M, and
c. a return type of Response<R> where R is the return type of M

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

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

Deleted: 01

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

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 shown in Snippet 11-1:

Deleted: 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>
```

Snippet 11-1: Example WSDL Interface

Formatted: Caption

The JAX-WS asynchronous mapping will produce the Java interface in Snippet 11-2:

Deleted: following

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

Snippet 11-2: JAX-WS Asynchronous Interface for WSDL Interface in Snippet 11-1

Formatted: Caption, Don't adjust space between Latin and Asian text

For SCA interface definition purposes, this is treated as equivalent to the interface in Snippet 11-3:

Deleted: following

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

Snippet 11-3: Equivalent SCA Interface Correspoining to Java Interface in Snippet 11-2

Formatted: Caption, Don't adjust space between Latin and Asian text

SCA runtimes MUST support the use of the JAX-WS client asynchronous model. [JCA100009] 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.

Deleted: In the above example, i

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]

Deleted: 01

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

3461 Asynchronous service methods are identified as described in the section "Asynchronous handling of Long
3462 Running Service Operations" and are mapped to WSDL in the same way as the equivalent synchronous
3463 method described in that section.

3464 Generating an asynchronous service method from a WSDL request/response operation follows the
3465 algorithm described in the same section.

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

Deleted: 01

12 Conformance

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

Normative code artifacts related to this specification are considered to be authoritative and take precedence over specification text.

There are three categories of artifacts for which this specification defines conformance:

a) SCA Java XML Document,

b) SCA Java Class

c) SCA Runtime.

12.1 SCA Java XML Document

An SCA Java XML document is an SCA Composite Document, or an SCA ComponentType Document, as defined by the SCA Assembly Model specification [ASSEMBLY], that uses the <interface.java> element. Such an SCA Java XML document MUST be a conformant SCA Composite Document or SCA ComponentType Document, as defined by the SCA Assembly Model specification [ASSEMBLY], and MUST comply with the requirements specified in the Interface section of this specification.

12.2 SCA Java Class

An SCA Java Class is a Java class or interface that complies with Java Standard Edition version 5.0 and MAY include annotations and APIs defined in this specification. An SCA Java Class that uses annotations and APIs defined in this specification MUST comply with the requirements specified in this specification for those annotations and APIs.

12.3 SCA Runtime

The APIs and annotations defined in this specification are meant to be used by Java-based component implementation models in either partial or complete fashion. A Java-based component implementation specification that uses this specification specifies which of the APIs and annotations defined here are used. The APIs and annotations an SCA Runtime has to support depends on which Java-based component implementation specification the runtime supports. For example, see the SCA POJO Component Implementation Specification [JAVA_CI].

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

1. The implementation MUST meet all the conformance requirements defined by the SCA Assembly Model Specification [ASSEMBLY].
2. The implementation MUST support <interface.java> and MUST comply with all the normative statements in Section 3.
3. The implementation MUST reject an SCA Java XML Document that does not conform to the sca-interface-java.xsd schema.
4. The implementation MUST support and comply with all the normative statements in Section 10.

Formatted: Indent: Before: 0 pt, Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 36 pt + Tab after: 54 pt + Indent at: 54 pt, Tabs: 18 pt, List tab + Not at 54

Deleted: 01

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>
```

Deleted: ¶

Deleted: 01

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
     */
    private SCAClientFactory() {
    }

    /**
     * Constructor used by concrete subclasses
```

```
3590      * @param domainURI - The Domain URI of the Domain accessed via this
3591      * SCAClientFactory
3592      */
3593     protected SCAClientFactory(URI domainURI) {
3594    throws NoSuchDomainException {
3595         this.domainURI = domainURI;
3596     }
3597
3598     /**
3599      * Gets the Domain URI of the Domain accessed via this SCAClientFactory
3600      * @return - the URI for the Domain
3601      */
3602     protected URI getDomainURI() {
3603         return domainURI;
3604     }
3605
3606
3607     /**
3608      * Creates a new instance of the SCAClient that can be
3609      * used to lookup SCA Services.
3610      *
3611      * @param domainURI      URI of the target domain for the SCAClient
3612      * @return A new SCAClient
3613      */
3614     public static SCAClientFactory newInstance( URI domainURI )
3615          throws NoSuchDomainException {
3616         return newInstance(null, null, domainURI);
3617     }
3618
3619     /**
3620      * Creates a new instance of the SCAClient that can be
3621      * used to lookup SCA Services.
3622      *
3623      * @param properties   Properties that may be used when
3624      * creating a new instance of the SCAClient
3625      * @param domainURI      URI of the target domain for the SCAClient
3626      * @return A new SCAClient instance
3627      */
3628     public static SCAClientFactory newInstance(Properties properties,
3629                                                            URI domainURI)
3630          throws NoSuchDomainException {
3631         return newInstance(properties, null, domainURI);
3632     }
3633
3634     /**
3635      * Creates a new instance of the SCAClient that can be
3636      * used to lookup SCA Services.
3637      *
3638      * @param classLoader   ClassLoader that may be used when
3639      * creating a new instance of the SCAClient
3640      * @param domainURI      URI of the target domain for the SCAClient
3641      * @return A new SCAClient instance
3642      */
3643     public static SCAClientFactory newInstance(ClassLoader classLoader,
3644                                                            URI domainURI)
3645          throws NoSuchDomainException {
3646         return newInstance(null, classLoader, domainURI);
3647     }
3648
3649     /**
3650      * Creates a new instance of the SCAClient that can be
3651      * used to lookup SCA Services.
3652      *
3653      * @param properties    Properties that may be used when
```

Deleted: 01

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

```
3654      * creating a new instance of the SCAClient
3655      * @param classLoader   ClassLoader that may be used when
3656      * creating a new instance of the SCAClient
3657      * @param domainURI      URI of the target domain for the SCAClient
3658      * @return A new SCAClient instance
3659      */
3660     public static SCAClientFactory newInstance(Properties properties,
3661                                          ClassLoader classLoader,
3662                                          URI domainURI)
3663          throws NoSuchDomainException {
3664         final SCAClientFactoryFinder finder =
3665             factoryFinder != null ? factoryFinder :
3666                 new SCAClientFactoryFinderImpl();
3667         final SCAClientFactory factory
3668             = finder.find(properties, classLoader, domainURI);
3669         return factory;
3670     }
3671
3672     /**
3673      * Returns a reference proxy that implements the business interface <T>
3674      * of a service in the SCA Domain handled by this SCAClientFactory
3675      *
3676      * @param serviceURI the relative URI of the target service. Takes the
3677      * form componentName/serviceName.
3678      * Can also take the extended form componentName/serviceName/bindingName
3679      * to use a specific binding of the target service
3680      *
3681      * @param interfaze The business interface class of the service in the
3682      * domain
3683      * @param <T> The business interface class of the service in the domain
3684      *
3685      * @return a proxy to the target service, in the specified SCA Domain
3686      * that implements the business interface <B>.
3687      * @throws NoSuchServiceException Service requested was not found
3688      * @throws NoSuchDomainException Domain requested was not found
3689      */
3690     public abstract <T> T getService(Class<T> interfaze, String serviceURI)
3691         throws NoSuchServiceException, NoSuchDomainException;
3692 }
```

3693
B.1.2 SCAClientFactoryFinder interface

3694 The SCAClientFactoryFinder interface is a Service Provider Interface representing a SCAClientFactory
3695 finder. SCA provides a default reference implementation of this interface. SCA runtime vendors can
3696 create alternative implementations of this interface that use different class loading or lookup mechanisms.

3697

```
3698 /*
3699  * Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
3700  * OASIS trademark, IPR and other policies apply.
3701  */
3702
3703 package org.oasisopen.sca.client;
3704
3705 import java.net.URI;
3706 import java.util.Properties;
3707
3708 import org.oasisopen.sca.NoSuchDomainException;
3709
3710 /* A Service Provider Interface representing a SCAClientFactory finder.
3711  * SCA provides a default reference implementation of this interface.
3712  * SCA runtime vendors can create alternative implementations of this
3713  * interface that use different class loading or lookup mechanisms.
3714  */
```

Deleted: 01

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

3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732

```
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 ;
}
```

3733
B.1.3 SCAClientFactoryFinderImpl class

3734 3735 3736
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.

3737 3738
It discovers a provider's SCAClientFactory implementation by referring to the following information in this order:

3739 3740
1. The org.oasisopen.sca.client.SCAClientFactory property from the Properties specified on the newInstance() method call if specified

3741
2. The org.oasisopen.sca.client.SCAClientFactory property from the System Properties

3742
3. The META-INF/services/org.oasisopen.sca.client.SCAClientFactory file

3743

Formatted: No bullets or numbering

3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773

```
/*
 * 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;
import java.io.Closeable;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.lang.reflect.Constructor;
import java.net.URI;
import java.net.URL;
import java.util.Properties;

import org.oasisopen.sca.NoSuchDomainException;
import org.oasisopen.sca.ServiceRuntimeException;
import org.oasisopen.sca.client.SCAClientFactory;

/**
 * This is a default implementation of an SCAClientFactoryFinder which is
 * used to find an implementation of the SCAClientFactory interface.
 *
 * @see SCAClientFactoryFinder
 * @see SCAClientFactory
 *
 * @author OASIS Open
```

Deleted: 01

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

```
3774     */
3775    public class SCAClientFactoryFinderImpl implements SCAClientFactoryFinder {
3776
3777        /**
3778         * The name of the System Property used to determine the SPI
3779         * implementation to use for the SCAClientFactory.
3780         */
3781        private static final String SCA_CLIENT_FACTORY_PROVIDER_KEY =
3782             SCAClientFactory.class.getName();
3783
3784        /**
3785         * The name of the file loaded from the ClassPath to determine
3786         * the SPI implementation to use for the SCAClientFactory.
3787         */
3788        private static final String SCA_CLIENT_FACTORY_PROVIDER_META_INF_SERVICE
3789            = "META-INF/services/" + SCA_CLIENT_FACTORY_PROVIDER_KEY;
3790
3791        /**
3792         * Public Constructor
3793         */
3794        public SCAClientFactoryFinderImpl() {
3795        }
3796
3797        /**
3798         * Creates an instance of the SCAClientFactorySPI implementation.
3799         * This discovers the SCAClientFactorySPI Implementation and instantiates
3800         * the provider's implementation.
3801         *
3802         * @param properties    Properties that may be used when creating a new
3803         * instance of the SCAClient
3804         * @param classLoader   ClassLoader that may be used when creating a new
3805         * instance of the SCAClient
3806         * @return new instance of the SCAClientFactory
3807         * @throws ServiceRuntimeException Failed to create SCAClientFactory
3808         * Implementation.
3809         */
3810        public SCAClientFactory find(Properties properties,
3811                                     ClassLoader classLoader,
3812                                     URI domainURI )
3813             throws NoSuchDomainException, ServiceRuntimeException {
3814             if (classLoader == null) {
3815                    classLoader = getThreadContextClassLoader ();
3816             }
3817             final String factoryImplClassName =
3818                    discoverProviderFactoryImplClass(properties, classLoader);
3819             final Class<? extends SCAClientFactory> factoryImplClass
3820                    = loadProviderFactoryClass(factoryImplClassName,
3821                                                        classLoader);
3822             final SCAClientFactory factory =
3823                    instantiateSCAClientFactoryClass(factoryImplClass,
3824                                                            domainURI );
3825             return factory;
3826       }
3827
3828        /**
3829         * Gets the Context ClassLoader for the current Thread.
3830         *
3831         * @return The Context ClassLoader for the current Thread.
3832         */
3833        private static ClassLoader getThreadContextClassLoader () {
3834            final ClassLoader threadClassLoader =
3835             Thread.currentThread().getContextClassLoader();
3836            return threadClassLoader;
3837        }
```

Deleted: 01

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

```
3838
3839        /**
3840         * Attempts to discover the class name for the SCAClientFactorySPI
3841         * implementation from the specified Properties, the System Properties
3842         * or the specified ClassLoader.
3843         *
3844         * @return The class name of the SCAClientFactorySPI implementation
3845         * @throw ServiceRuntimeException Failed to find implementation for
3846         * SCAClientFactorySPI.
3847         */
3848        private static String
3849            discoverProviderFactoryImplClass(Properties properties,
3850                                                 ClassLoader classLoader)
3851            throws ServiceRuntimeException {
3852            String providerClassName =
3853             checkPropertiesForSPIClassName(properties);
3854            if (providerClassName != null) {
3855                return providerClassName;
3856            }
3857
3858            providerClassName =
3859             checkPropertiesForSPIClassName(System.getProperties());
3860            if (providerClassName != null) {
3861                return providerClassName;
3862            }
3863
3864            providerClassName = checkMETAINFServicesForSIPClassName(classLoader);
3865            if (providerClassName == null) {
3866                throw new ServiceRuntimeException(
3867                    "Failed to find implementation for SCAClientFactory");
3868            }
3869
3870            return providerClassName;
3871        }
3872
3873        /**
3874         * Attempts to find the class name for the SCAClientFactorySPI
3875         * implementation from the specified Properties.
3876         *
3877         * @return The class name for the SCAClientFactorySPI implementation
3878         * or <code>null</code> if not found.
3879         */
3880        private static String
3881            checkPropertiesForSPIClassName(Properties properties) {
3882            if (properties == null) {
3883                return null;
3884            }
3885
3886            final String providerClassName =
3887             properties.getProperty(SCA_CLIENT_FACTORY_PROVIDER_KEY);
3888            if (providerClassName != null && providerClassName.length() > 0) {
3889                return providerClassName;
3890            }
3891
3892            return null;
3893        }
3894
3895        /**
3896         * Attempts to find the class name for the SCAClientFactorySPI
3897         * implementation from the META-INF/services directory
3898         *
3899         * @return The class name for the SCAClientFactorySPI implementation or
3900         * <code>null</code> if not found.
3901         */
```

Deleted: 01

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

```
3902        private static String checkMETAINFServicesForSIPClassName(ClassLoader cl)
3903        {
3904            final URL url =
3905              cl.getResource(SCA_CLIENT_FACTORY_PROVIDER_META_INF_SERVICE);
3906            if (url == null) {
3907                return null;
3908            }
3909
3910            InputStream in = null;
3911            try {
3912                in = url.openStream();
3913                BufferedReader reader = null;
3914                try {
3915                    reader =
3916                          new BufferedReader(new InputStreamReader(in, "UTF-8"));
3917
3918                    String line;
3919                    while ((line = readNextLine(reader)) != null) {
3920                        if (!line.startsWith("#") && line.length() > 0) {
3921                            return line;
3922                        }
3923                    }
3924
3925                    return null;
3926                } finally {
3927                    closeStream(reader);
3928                }
3929            } catch (IOException ex) {
3930                throw new ServiceRuntimeException(
3931                          "Failed to discover SCAClientFactory provider", ex);
3932            } finally {
3933                closeStream(in);
3934            }
3935        }
3936
3937        /**
3938         * Reads the next line from the reader and returns the trimmed version
3939         * of that line
3940         *
3941         * @param reader The reader from which to read the next line
3942         * @return The trimmed next line or <code>null</code> if the end of the
3943         * stream has been reached
3944         * @throws IOException I/O error occurred while reading from Reader
3945         */
3946        private static String readNextLine(BufferedReader reader)
3947            throws IOException {
3948
3949            String line = reader.readLine();
3950            if (line != null) {
3951                line = line.trim();
3952            }
3953            return line;
3954        }
3955
3956        /**
3957         * Loads the specified SCAClientFactory Implementation class.
3958         *
3959         * @param factoryImplClassName The name of the SCAClientFactory
3960         * Implementation class to load
3961         * @return The specified SCAClientFactory Implementation class
3962         * @throws ServiceRuntimeException Failed to load the SCAClientFactory
3963         * Implementation class
3964         */
3965        private static Class<? extends SCAClientFactory>
```

Deleted: 01

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

```
3966            loadProviderFactoryClass(String factoryImplClassName,
3967                                     ClassLoader classLoader)
3968            throws ServiceRuntimeException {
3969
3970            try {
3971                final Class<?> providerClass =
3972                    classLoader.loadClass(factoryImplClassName);
3973                final Class<? extends SCAClientFactory> providerFactoryClass =
3974                    providerClass.asSubclass(SCAClientFactory.class);
3975                return providerFactoryClass;
3976            } catch (ClassNotFoundException ex) {
3977                throw new ServiceRuntimeException(
3978                    "Failed to load SCAClientFactory implementation class "
3979                    + factoryImplClassName, ex);
3980            } catch (ClassCastException ex) {
3981                throw new ServiceRuntimeException(
3982                           "Loaded SCAClientFactory implementation class "
3983                           + factoryImplClassName
3984                        + " is not a subclass of "
3985                        + SCAClientFactory.class.getName() , ex);
3986            }
3987        }
3988
3989        /**
3990         * Instantiate an instance of the specified SCAClientFactorySPI
3991         * Implementation class.
3992         *
3993         * @param factoryImplClass The SCAClientFactorySPI Implementation
3994         * class to instantiate.
3995         * @return An instance of the SCAClientFactorySPI Implementation class
3996         * @throws ServiceRuntimeException Failed to instantiate the specified
3997         * specified SCAClientFactorySPI Implementation class
3998         */
3999        private static SCAClientFactory instantiateSCAClientFactoryClass(
4000                           Class<? extends SCAClientFactory> factoryImplClass,
4001                    URI domainURI)
4002            throws NoSuchDomainException, ServiceRuntimeException {
4003
4004            try {
4005                Constructor<? extends SCAClientFactory> URIConstructor =
4006                    factoryImplClass.getConstructor(domainURI.getClass());
4007                SCAClientFactory provider =
4008                   URIConstructor.newInstance( domainURI );
4009                return provider;
4010            } catch (Throwable ex) {
4011                throw new ServiceRuntimeException(
4012                   "Failed to instantiate SCAClientFactory implementation class "
4013                   + factoryImplClass, ex);
4014            }
4015        }
4016
4017        /**
4018         * Utility method for closing Closeable Object.
4019         *
4020         * @param closeable The Object to close.
4021         */
4022        private static void closeStream(Closeable closeable) {
4023            if (closeable != null) {
4024                try{
4025                    closeable.close();
4026                } catch (IOException ex) {
4027                    throw new ServiceRuntimeException("Failed to close stream",
4028                                                            ex);
4029                }
```

Deleted: 01

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

```
            }
        }
    }
```

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 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:

Deleted: ¶
¶

Deleted: ¶

Deleted: 01

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

4058 # C. Conformance Items

4059 This section contains a list of conformance items for the SCA-J Common Annotations and APIs
4060 specification.

4061

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

Deleted: 01

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

[JCA30003]	if the Java interface class identified by the @interface attribute does contain a Java @Callback annotation, then the Java interface class identified by the @callbackInterface attribute MUST be the same interface class.
[JCA30004]	The interface.java element MUST conform to the schema defined in the sca-interface-java.xsd schema.
[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

Deleted: 01

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

	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 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.

Deleted: 01

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

[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.
[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

Formatted: Font color: Red

Formatted: Font color: Red

Deleted: 01

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

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:

- void return type
- a method name with the suffix "Async"
- a last input parameter with a type of ResponseDispatch<X>
- annotation with the asyncInvocation intent
- 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 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

Formatted: Font color: Red

Deleted: 01

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

	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.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[JCA80003]	When invoked during the execution of a service operation, the RequestContext.getServiceReference method MUST return a ServiceReference that represents the service that was invoked.	**Formatted:** Pattern: Clear (Yellow)
[JCA80004]	The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if the reference named by the referenceName parameter has multiplicity greater than one.	**Formatted:** Font: Arial, Pattern: Clear (Yellow)
[JCA80005]	The ComponentContext.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.	**Formatted:** Font: Arial, Pattern: Clear (Yellow)
[JCA80006]	The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if the component does not have a reference with the name provided in the referenceName parameter.	**Formatted:** Font: Arial, Pattern: Clear (Yellow)
[JCA80007][JCA80007]	The ComponentContext.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.	**Formatted:** Font: Arial, Pattern: Clear (Yellow)
[JCA80008]	The ComponentContext.getURI method MUST return the absolute URI of the component in the SCA Domain.	**Formatted:** Pattern: Clear (Yellow)
[JCA80009]	The ComponentContext.getService method MUST return the proxy object implementing the interface provided by the businessInterface parameter, for the reference named by the referenceName parameter with the interface defined by the businessInterface parameter when that reference has a target service configured.	**Formatted:** Pattern: Clear (Yellow)
[JCA80010]	The ComponentContext.getService 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.	**Formatted:** Pattern: Clear (Yellow)
[JCA80011]	The ComponentContext.getService method MUST throw an IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter.	**Formatted:** Pattern: Clear (Yellow)

Deleted: 01

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

[JCA80012]	The ComponentContext.getService method MUST throw an IllegalArgumentException if the service reference with the name supplied in the referenceName does not have an interface compatible with the interface supplied in the businessInterface parameter.
[JCA80013]	The ComponentContext.getServiceReference method MUST return a ServiceReference object typed by the interface provided by the businessInterface parameter, for the reference named by the referenceName parameter with the interface defined by the businessInterface parameter when that reference has a target service configured.
[JCA80014]	The ComponentContext.getServices method MUST return a collection containing one proxy object implementing the interface provided by the businessInterface parameter for each of the target services configured on the reference identified by the referenceName parameter.
[JCA80015]	The ComponentContext.getServices method MUST return an empty collection if the service reference with the name supplied in the referenceName parameter is not wired to any target services.
[JCA80016]	The ComponentContext.getServices method MUST throw an IllegalArgumentException if the reference identified by the referenceName parameter has multiplicity of 0..1 or 1..1.
[JCA80017]	The ComponentContext.getServices method MUST throw an IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter.
The ComponentContext.getServices method MUST throw an IllegalArgumentException if the service reference with the name supplied in the referenceName does not have an interface compatible with the interface supplied in the businessInterface parameter.[JCA80018]	The ComponentContext.getServices method MUST throw an IllegalArgumentException if the service reference with the name supplied in the referenceName does not have an interface compatible with the interface supplied in the businessInterface parameter.
[JCA80019]	The ComponentContext.getServiceReferences method MUST return a collection containing one ServiceReference object typed by the interface provided by the businessInterface parameter for each of the target services configured on the reference identified by the referenceName parameter.
[JCA80020]	The ComponentContext.getServiceReferences method MUST return an empty collection if the service reference with the name supplied in the referenceName parameter is not wired to any target services.
[JCA80021]	The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the reference identified by the referenceName parameter has multiplicity of 0..1 or 1..1.
[JCA80022]	The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter.
[JCA80023]	The ComponentContext.getServiceReferences method MUST throw an

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

	IllegalArgumentException if the service reference with the name supplied in the referenceName does not have an interface compatible with the interface supplied in the businessInterface parameter.
[JCA80024]	The ComponentContext.createSelfReference method MUST return a ServiceReference object typed by the interface defined by the businessInterface parameter for one of the services of the invoking component which has the interface defined by the businessInterface parameter.
[JCA80025]	The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the component does not have a service which implements the interface identified by the businessInterface parameter.
[JCA80026]	The ComponentContext.createSelfReference method MUST return a ServiceReference object typed by the interface defined by the businessInterface parameter for the service identified by the serviceName of the invoking component and which has the interface defined by the businessInterface parameter.
[JCA80027]	The ComponentContext.createSelfReference method MUST throw an IllegalArgumentException if the component does not have a service with the name identified by the serviceName parameter.
[JCA80028]	The ComponentContext.createSelfReference method MUST throw an IllegalArgumentException if the component service with the name identified by the serviceName parameter does not implement a business interface which is compatible with the supplied businessInterface parameter.
[JCA80029]	The ComponentContext.getProperty method MUST return an object of the type identified by the type parameter containing the value specified in the component configuration for the property named by the propertyName parameter or null if no value is specified in the configuration.
[JCA80030]	The ComponentContext.getProperty method MUST throw an IllegalArgumentException if the component does not have a property with the name identified by the propertyName parameter.
[JCA80031]	The ComponentContext.getProperty method MUST throw an IllegalArgumentException if the component property with the name identified by the propertyName parameter does not have a type which is compatible with the supplied type parameter.
[JCA80032]	The ComponentContext.cast method MUST return a ServiceReference object which is typed by the same business interface as specified by the reference proxy object supplied in the target parameter.
[JCA80033]	The ComponentContext.cast method MUST throw an IllegalArgumentException if the supplied target parameter is not an SCA reference proxy object.
[JCA80034]	The RequestContext.getSecuritySubject method MUST return the JAAS subject of the current request, or null if there is no subject or null if the method is invoked from code not processing a service request or callback request.
[JCA80035]	The RequestContext.getServiceName method MUST return the name of the service for which an operation is being processed, or null if

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

	invoked from a thread that is not processing a service operation or a callback operation.	
[JCA80036]	The RequestContext.getCallbackReference method MUST return a ServiceReference object typed by the interface of the callback supplied by the client of the invoked service, or null if either the invoked service is not bidirectional or if the method is invoked from a thread that is not processing a service operation.	**Formatted:** Pattern: Clear (Yellow)
[JCA80037]	The RequestContext.getCallback method MUST return a reference proxy object typed by the interface of the callback supplied by the client of the invoked service, or null if either the invoked service is not bidirectional or if the method is invoked from a thread that is not processing a service operation.	**Formatted:** Pattern: Clear (Yellow)
[JCA80038]	When invoked during the execution of a callback operation, the RequestContext.getServiceReference method MUST return a ServiceReference that represents the callback that was invoked.	**Formatted:** Pattern: Clear (Yellow)
[JCA80039]	When invoked from a thread not involved in the execution of either a service operation or of a callback operation, the RequestContext.getServiceReference method MUST return null.	**Formatted:** Pattern: Clear (Yellow)
[JCA80040]	The ServiceReference.getService method MUST return a reference proxy object which can be used to invoke operations on the target service of the reference and which is typed with the business interface of the reference.	**Formatted:** Pattern: Clear (Yellow)
[JCA80041]	The ServiceReference.getBusinessInterface method MUST return a Class object representing the business interface of the reference.	**Formatted:** Pattern: Clear (Yellow)
[JCA80042]	The SCAClientFactory.newInstance(URI) method MUST return an object which implements the SCAClientFactory class for the SCA Domain identified by the domainURI parameter.	**Formatted:** Pattern: Clear (Yellow)
[JCA80043]	The SCAClientFactory.newInstance(URI) method MUST throw a NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.	**Formatted:** Pattern: Clear (Yellow)
[JCA80044]	The SCAClientFactory.newInstance(Properties, URI) method MUST return an object which implements the SCAClientFactory class for the SCA Domain identified by the domainURI parameter.	**Formatted:** Pattern: Clear (Yellow)
[JCA80045]	The SCAClientFactory.newInstance(Properties, URI) method MUST throw a NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.	**Formatted:** Pattern: Clear (Yellow)
[JCA80046]	The SCAClientFactory.newInstance(Classloader, URI) method MUST return an object which implements the SCAClientFactory class for the SCA Domain identified by the domainURI parameter.	**Formatted:** Pattern: Clear (Yellow)
[JCA80047]	The SCAClientFactory.newInstance(Classloader, URI) method MUST throw a NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.	**Formatted:** Pattern: Clear (Yellow)
[JCA80048]	The SCAClientFactory.newInstance(Properties, Classloader, URI) method MUST return an object which implements the SCAClientFactory class for the SCA Domain identified by the domainURI parameter.	**Formatted:** Pattern: Clear (Yellow)
[JCA80049]	The SCAClientFactory.newInstance(Properties, Classloader, URI)	**Formatted:** Pattern: Clear (Yellow)

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

	MUST throw a NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.	
[JCA80050]	The SCAClientFactory.getService method MUST return a proxy object which implements the business interface defined by the interfaze parameter and which can be used to invoke operations on the service identified by the serviceURI parameter.	**Formatted:** Pattern: Clear (Yellow)
[JCA80051]	The SCAClientFactory.getService method MUST throw a NoSuchServiceException 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.	**Formatted:** Pattern: Clear (Yellow)
[JCA80052]	The SCAClientFactory.getService method MUST throw a NoSuchServiceException if the domainURI of the SCAClientFactory does not identify a valid SCA Domain.	**Formatted:** Pattern: Clear (Yellow)
[JCA80053]	The SCAClientFactory.getDomainURI method MUST return the SCA Domain URI of the Domain associated with the SCAClientFactory object.	**Formatted:** Pattern: Clear (Yellow)
[JCA80054]	The SCAClientFactory.getDomainURI method MUST throw a ***NoSuchServiceException*** if the domainURI of the SCAClientFactory does not identify a valid SCA Domain.	**Formatted:** Pattern: Clear (Yellow)
The implementation of the SCAClientFactoryFinder.find method MUST return an object which is an implementation of the SCAClientFactory interface, for the SCA Domain represented by the doaminURI parameter, using the supplied properties and classloader. [JCA80055]	The implementation of the SCAClientFactoryFinder.find method MUST return an object which is an implementation of the SCAClientFactory interface, for the SCA Domain represented by the doaminURI parameter, using the supplied properties and classloader.	**Formatted:** Pattern: Clear (Yellow) **Formatted:** Pattern: Clear (Yellow)
[JCA80056]	The implementation of the SCAClientFactoryFinder.find method MUST throw a ServiceRuntimeException if the SCAClientFactory implementation could not be found.	**Formatted:** Pattern: Clear (Yellow)
[JCA50057]	The ResponseDispatch.sendResponse() method MUST send the response message to the client of an asynchronous service.	**Formatted:** Pattern: Clear (Yellow)
[JCA80058]	The ResponseDispatch.sendResponse() method MUST throw an InvalidStateException if either the sendResponse method or the sendFault method has already been called once.	**Formatted:** Pattern: Clear (Yellow)
[JCA80059]	The ResponseDispatch.sendFault() method MUST send the supplied fault to the client of an asynchronous service.	**Formatted:** Pattern: Clear (Yellow)
[JCA80060]	The ResponseDispatch.sendFault() method MUST throw an InvalidStateException if either the sendResponse method or the sendFault method has already been called once.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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	**Formatted:** Pattern: Clear (Yellow)

Deleted: 01

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

	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.	**Formatted:** Pattern: Clear (Yellow)
[JCA90004]	A method annotated with @Destroy can have any access modifier and MUST have a void return type and no arguments.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[JCA90007]	When marked for eager initialization with an @EagerInit annotation, the composite scoped instance MUST be created when its containing component is started.	**Formatted:** Pattern: Clear (Yellow)
[JCA90008]	A method marked with the @Init annotation can have any access modifier and MUST have a void return type and no arguments.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[JCA90011]	The @Property annotation MUST NOT be used on a class field that is declared as final.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[JCA90014]	For a @Property annotation applied to a constructor parameter, the required attribute MUST NOT have the value false.	**Formatted:** Pattern: Clear (Yellow)
[JCA90015]	The @Qualifier annotation MUST be used in a specific intent annotation definition where the intent has qualifiers.	**Formatted:** Pattern: Clear (Yellow)
[JCA90016]	The @Reference annotation MUST NOT be used on a class field that is declared as final.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[JCA90019]	For a @Reference annotation applied to a constructor parameter, the required attribute MUST have the value true.	**Formatted:** Pattern: Clear (Yellow)
[JCA90020]	If the type of a reference is not an array or any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference/> element with @multiplicity= 0..1 if the @Reference annotation required attribute is false and with @multiplicity=1..1 if the @Reference annotation required attribute is true.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)

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

[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).	**Formatted:** Pattern: Clear (Yellow)
[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).	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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.	**Formatted:** Pattern: Clear (Yellow)
[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	**Formatted:** Pattern: Clear (Yellow)

Deleted: 01

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

	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

Formatted: Pattern: Clear (Yellow)

Deleted: 01

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

[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 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.
[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.
[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.
[JCA100005]	SCA runtimes MAY support the SDO 2.1 mapping from XML schema types to Java and from Java to XML Schema.
[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

Formatted: Font color: Red

Formatted: Font color: Red

Formatted: Font color: Red

Formatted: Font color: Red

Deleted: 01

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

	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.
[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.

4062

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

4063 # D. Acknowledgements

4064 The following individuals have participated in the creation of this specification and are gratefully
4065 acknowledged:

4066 **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: 01

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

Scott Vorthmann	TIBCO Software Inc.
Feng Wang	Primeton Technologies, Inc.
Robin Yang	Primeton Technologies, Inc.

4067

Deleted: ¶
¶

<#>Non-Normative Text¶

Formatted: Bullets and Numbering

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

Deleted: 01

4068

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

Deleted: 01

4069
E. Revision History

4070 [optional; should not be included in OASIS Standards]

4071

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

Deleted: 01

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: 01

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
cd03-rev3	2010-02-01	Mike Edwards	Applied issue 54. Editorial updates to code samples.
cd03-rev4	2010-02-05	Bryan Aupperle, Dave Booz	Editorial update for OASIS formatting

4072

Deleted: 01

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