

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

Committee Draft 04

06 February 2010

Specification URIs:

This Version:

http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd04.html

http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd04.doc

http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd04.pdf (Authoritative)

Previous 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)

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)

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

Compiled Java API:

http://docs.oasis-open.org/opencsa/sca-j/sca-caa-apis-1.1-CD04.jar

Downloadable Javadoc:

http://docs.oasis-open.org/opencsa/sca-j/sca-j-caa-javadoc-1.1-CD04.zip

Hosted Javadoc:

http://docs.oasis-open.org/opencsa/sca-j/javadoc/index.html

Declared XML Namespace(s):

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

Java Artifacts:

http://docs.oasis-open.org/opencsa/sca-j/sca-j-common-annotations-and-apis-1.1-cd04.zip

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.oasisopen.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.oasisopen.org/committees/sca-j/ipr.php.

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

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 name, "OASIS," is a 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: s

Deleted: 2009

Deleted: "

DisplayText cannot span mo

sca-javacaa-1.1-spec-<u>cd04</u> Copyright © OASIS® 2005, 2010. All Rights Reserved. 06 Feb 2010 Page 3 of 126

Table of Contents

1	Introduction	<u></u> 7
	1.1 Terminology	<u></u> 7
	1.2 Normative References.	<u></u> 7
	1.3 Non-Normative References	<u></u> 8
2	Implementation Metadata	<u></u> g
	2.1 Service Metadata	<u></u> g
	2.1.1 @ Service	<u></u> g
	2.1.2 Java Semantics of a Remotable Service	<u></u> g
	2.1.3 Java Semantics of a Local Service	<u></u> g
	2.1.4 @Reference	<u></u> 10
	2.1.5 @ Property	
	2.2 Implementation Scopes: @Scope, @Init, @Destroy	<u></u> 10
	2.2.1 Stateless Scope	<u></u> 10
	2.2.2 Composite Scope.	<u></u> 11
	2.3 @ Allows Pass By Reference	
	2.3.1 Marking Services as "allows pass by reference"	<u></u> 12
	2.3.2 Marking References as "allows pass by reference"	
	2.3.3 Applying "allows pass by reference" to Service Proxies	
	2.3.4 Using "allows pass by reference" to Optimize Remotable Calls	<u></u> 13
3	Interface	
	3.1 Java Interface Element – <interface.java></interface.java>	
	3.2 @Remotable	
	3.3 @Callback	
	3.4 @ Asynchrvocation	
	3.5 SCA Java Annotations for Interface Classes	
	3.6 Compatibility of Java Interfaces.	
<u>4</u>	SCA Component Implementation Lifecycle	
	4.1 Overview of SCA Component Implementation Lifecycle	
	4.2 SCA Component Implementation Lifecycle State Diagram	
	4.2.1 Constructing State	
	4.2.2 Injecting State	
	4.2.3 Initializing State	
	4.2.4 Running State	
	4.2.5 Destroying State	
_	4.2.6 Terminated State	
5	Client API	
	5.1 Accessing Services from an SCA Component	
	5.1.1 Using the Component Context API	
	5.2 Accessing Services from non-SCA Component Implementations	
_	5.2.1 SCAClientFactory Interface and Related Classes	
<u>6</u>		
<u>7</u>	7, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,	
	7.1 @OneWay	<u></u> 24

7.2 Callbacks	24
7.2.1 Using Callbacks	24
7.2.2 Callback Instance Management	<u></u> 26
7.2.3 Callback Injection	<u></u> 26
7.2.4 Implementing Multiple Bidirectional Int	<u>erfaces</u> 26
7.2.5 Accessing Callbacks	27
7.3 Asynchronous handling of Long Running S	ervice Operations28
7.4 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 Annotati	<u>ons</u> 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	<u>Intents</u> 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
	47
	48
	50
	51
· · · · · · · · · · · · · · · · · · ·	52
· · · · ·	52
-	52
	53
· · · · ·	53
	57
	58
	59
	59
	60
· · · · · · · · · · · · · · · · · · ·	60
	<u></u> 61
	62
	62
	63
	63
	65
	65
· · · · ·	66
10.10 @Context	<u></u> 67

<u>1</u>	10.11 @Destroy	68
	10.12 @EagerInit	
<u>1</u>	10.13 @Init	<u></u> 69
<u>1</u>	10.14 @Integrity	70
<u>1</u>	10.15 @Intent	71
1	10.16 @ManagedSharedTransaction	71
<u>1</u>	10.17 @ManagedTransaction	72
<u>1</u>	10.18 @MutualAuthentication	73
<u>1</u>	10.19 @NoManagedTransaction	73
<u>1</u>	10.20 @OneWay	74
<u>1</u>	10.21 @PolicySets	75
<u>1</u>	10.22 @Property	75
<u>1</u>	10.23 @Qualifier	77
<u>1</u>	10.24 @Reference	78
	10.24.1 Reinjection	80
<u>1</u>	10.25 @Remotable	82
<u>1</u>	10.26 @Requires	<u></u> 84
<u>1</u>	10.27 @Scope	<u></u> 84
<u>1</u>	10.28 @Service	<u></u> 85
11	WSDL to Java and Java to WSDL	<u></u> 87
<u>1</u>	11.1 JAX-WS Annotations and SCA Interfaces	<u></u> 87
<u>1</u>	11.2 JAX-WS Client Asynchronous API for a Synchronous Service	<u></u> 92
<u>1</u>	11.3 Treatment of SCA Asynchronous Service API	<u></u> 93
12	Conformance	<u></u> 95
<u>1</u>	12.1 SCA Java XML Document	<u></u> 95
<u>1</u>	12.2 SCA Java Class	<u></u> 95
<u>1</u>	12.3 SCA Runtime	<u></u> 95
<u>A.</u>	XML Schema: sca-interface-java-1.1.xsd	<u></u> 96
В.	Java Classes and Interfaces	<u></u> 97
<u>E</u>	B.1 SCAClient Classes and Interfaces	<u></u> 97
	B.1.1 SCAClientFactory Class	<u></u> 97
	B.1.2 SCAClientFactoryFinder interface	<u></u> 99
	B.1.3 SCAClientFactoryFinderImpl class	100
	B.1.4 SCAClient Classes and Interfaces - what does a vendor need to do?	105
<u>C.</u>	Conformance Items	106
D.	Acknowledgements	121
F	Revision History	124

1 Introduction

1

5

6 7

8

9

10

11 12

14

15

17

21

The SCA-J Common Annotations and APIs specification defines a Java syntax for programming concepts 2 3 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. 4

Specifically, this specification covers:

1. Jmplementation metadata for specifying component services, references, and properties

Deleted: ¶

- 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

13 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

16 complete fashion.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD 18

NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. 19

20

1.2 Normative References

22 23	[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.	
24 25	[ASSEMBLY]	OASIS, Committee Draft 05, "SCA Assembly Model Specification Version 1.1", January 2010.	Deleted: ,
26		http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-	Field Code Changed
27		<u>cd05</u> .pdf	Deleted: cd03
28 29	[JAVA_CI]	OASIS, Committee Draft 02, "SCA POJO Component Implementation Specification Version 1.1", February 2010.	
30		http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-cd02.pdf	Field Code Changed
31	[SDO]	SDO 2.1 Specification,	Deleted: cd01
32		http://www.osoa.org/download/attachments/36/Java-SDO-Spec-v2.1.0-FINAL.pdf	
33	[JAX-B]	JAXB 2.1 Specification,	
34		http://www.jcp.org/en/jsr/detail?id=222	
35	[WSDL]	WSDL Specification,	
36		WSDL 1.1: http://www.w3.org/TR/wsdl,	
37	[POLICY]	OASIS, Committee Draft 02, "SCA Policy Framework Version 1.1,", February	Deleted: ,
38		<u>2009.</u>	
39		http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.pdf	
40	[JSR-250]	Common Annotations for the Java Platform specification (JSR-250),	
41		http://www.jcp.org/en/jsr/detail?id=250	
42	[JAX-WS]	JAX-WS 2.1 Specification (JSR-224),	
43		http://www.jcp.org/en/jsr/detail?id=224	
44	[JAVABEANS]	JavaBeans 1.01 Specification,	
45		http://java.sun.com/javase/technologies/desktop/javabeans/api/	

46 47 48	[JAAS]	Java Authentication and Authorization Service Reference Guide http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html
49	1.3 Non-Norm	native References
50 51	[EBNF-Syntax]	Extended BNF syntax format used for formal grammar of constructs http://www.w3.org/TR/2004/REC-xml-20040204/#sec-notation

2 Implementation Metadata

53 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

52

54

55

56 57

60 61

62 63

64

65

66

67

68 69

70

71

72

73 74

75

76

77 78

79

80 81

82

83

84 85

86 87

88 89

90

91

 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]

<u>Snippet</u> 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

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

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.

sca-javacaa-1.1-spec-<u>cd04</u> Copyright © OASIS® 2005, 2010. All Rights Reserved. 06 Feb 2010 Page 9 of 126

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.

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

106 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

Snippet 2-3: Java Component Implementation with Lifecycle Methods

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.

sca-javacaa-1.1-spec-<u>cd04</u> Copyright © OASIS® 2005, 2010. All Rights Reserved. 06 Feb 2010 Page 10 of 126

- 136 The concurrency model for the stateless scope is single threaded. This means that the SCA runtime
- 137 MUST ensure that a stateless scoped implementation instance object is only ever dispatched on one
- 138 thread at any one time. [JCA20002] In addition, within the SCA lifecycle of a stateless scoped
- implementation instance, the SCA runtime MUST only make a single invocation of one business method. 139
- 140 [JCA20003] Note that the SCA lifecycle might not correspond to the Java object lifecycle due to runtime
- 141 techniques such as pooling.

142

148

149

150

171

2.2.2 Composite Scope

- 143 The meaning of "composite scope" is defined in relation to the composite containing the component.
- 144 It is important to distinguish between different uses of a composite, where these uses affect the numbers 145 of instances of components within the composite. There are 2 cases:
- 146 Where the composite containing the component using the Java implementation is the SCA Domain (i.e. a deployment composite declares the component using the implementation) 147
 - Where the composite containing the component using the Java implementation is itself used as the implementation of a higher level component (any level of nesting is possible, but the component is NOT at the Domain level)
- 151 Where an implementation is used by a "domain level component", and the implementation is marked 152 "Composite" scope, the SCA runtime MUST ensure that all consumers of the component appear to be interacting with a single runtime instance of the implementation. [JCA20004] 153
- 154 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 155
- MUST ensure that all consumers of the component appear to be interacting with a single runtime instance 156
- of the implementation. There can be multiple instances of the higher level component, each running on 157 158 different nodes in a distributed SCA runtime. [JCA20008]
- 159 The SCA runtime can exploit shared state technology in combination with other well known high
- 160 availability techniques to provide the appearance of a single runtime instance for consumers of composite 161 scoped components.
- 162 The lifetime of the containing composite is defined as the time it becomes active in the runtime to the time 163 it is deactivated, either normally or abnormally.
- 164 When the implementation class is marked for eager initialization, the SCA runtime MUST create a
- 165 composite scoped instance when its containing component is started. [JCA20005] If a method of an
- implementation class is marked with the @Init annotation, the SCA runtime MUST call that method when 166
- the implementation instance is created. [JCA20006] 167
- 168 The concurrency model for the composite scope is multi-threaded. This means that the SCA runtime MAY
- 169 run multiple threads in a single composite scoped implementation instance object and the SCA runtime
- 170 MUST NOT perform any synchronization. [JCA20007]

2.3 @AllowsPassByReference

- 172 Calls to remotable services (see section "Java Semantics of a Remotable Service") have by-value
- 173 semantics. This means that input parameters passed to the service can be modified by the service
- without these modifications being visible to the client. Similarly, the return value or exception from the 174
- 175 service can be modified by the client without these modifications being visible to the service
- 176 implementation. For remote calls (either cross-machine or cross-process), these semantics are a
- 177 consequence of marshalling input parameters, return values and exceptions "on the wire" and
- 178 unmarshalling them "off the wire" which results in physical copies being made. For local method calls
- 179 within the same JVM, Java language calling semantics are by-reference and therefore do not provide the
- 180 correct by-value semantics for SCA remotable interfaces. To compensate for this, the SCA runtime can
- 181 intervene in these calls to provide by-value semantics by making copies of any mutable objects passed.
- The cost of such copying can be very high relative to the cost of making a local call, especially if the data 182
- 183 being passed is large. Also, in many cases this copying is not needed if the implementation observes
- certain conventions for how input parameters, return values and exceptions are used. The 184
- @AllowsPassByReference annotation allows service method implementations and client references to be 185

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.

Deleted: and References

2.3.1 Marking Services as "allows pass by reference"

186

187

188

189 190

191

192

193

194 195

196

197

198

199 200

201

202

203 204

205 206

207

208

209 210

211

212 213

214 215

216

217

218

219

220

221

222

223

224

225

226

227 228

229

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".

Deleted: value Deleted: below

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".

Deleted: " (see definition below).

sca-iavacaa-1.1-spec-cd04 Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 12 of 126

2.3.4 Using "allows pass by reference" to Optimize Remotable Calls

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

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

sca-javacaa-1.1-spec-<u>cd04</u>
Copyright © OASIS® 2005, <u>2010.</u> All Rights Reserved.

3 Interface

239 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]

<u>Snippet</u> 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"?/>
```

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

283 on the <interface.java/> element does not override the presence of a @Remotable annotation on the 284 interface class and so if the interface class contains a @Remotable annotation and the @remotable 285 attribute has a value of "false", then the SCA Runtime MUST raise an error and MUST NOT run the component concerned. [JCA30005] 286 287 Deleted: The following snippet 288 <u>Snippet</u> 3-2 shows an example of the Java interface element: 289 290 <interface.java interface="services.stockquote.StockQuoteService"</pre> 291 callbackInterface="services.stockquote.StockQuoteServiceCallback"/> 292 Snippet 3-2 Example interface.java Element 293 294 Here, the Java interface is defined in the Java class file 295 ./services/stockquote/StockQuoteService.class, where the root directory is defined by the contribution 296 in which the interface exists. Similarly, the callback interface is defined in the Java class file ./services/stockquote/StockQuoteServiceCallback.class. 297 298 Note that the Java interface class identified by the @interface attribute can contain a Java @Callback 299 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 300 301 does contain a Java @Callback annotation, then the Java interface class identified by the @callbackInterface attribute MUST be the same interface class. [JCA30003] 302 303 For the Java interface type system, parameters and return types of the service methods are described 304 using Java classes or simple Java types. It is recommended that the Java Classes used conform to the 305 requirements of either JAXB [JAX-B] or of Service Data Objects [SDO] because of their integration with XML technologies. 306 3.2 @Remotable 307 308 The @Remotable annotation on a Java interface, a service implementation class, or a service reference Deleted: indicates that the 309 denotes an interface or class that is designed to be used for remote communication. Remotable 310 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 311 overloading. 312 3.3 @Callback 313 314 A callback interface is declared by using a @Callback annotation on a Java service interface, with the 315 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 316 implementation. 317 3.4 @AsyncInvocation 318

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 @Asynchronous 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.

sca-javacaa-1.1-spec-<u>cd04</u> Copyright © OASIS® 2005, 2010. All Rights Reserved.

319 320

321

322 323

324

325 326

327

328

06 Feb 2010 Page 15 of 126

329 330 331 332	For a service implementation, SCA provides an asynchronous service mapping of the WSDL request/response interface which enables the service implementation to send the response message at an arbitrary time after the original service operation is invoked. This is described in the section "Asynchronous handling of Long Running Service Operations".
333	3.5 SCA Java Annotations for Interface Classes
334 335	A Java interface referenced by the @interface attribute of an <interface.java></interface.java> element MUST NOT contain the following SCA Java annotations:
336	@Intent, @Qualifier. [JCA30008]
337 338	A Java interface referenced by the @interface attribute of an <interface.java></interface.java> element MUST NOT contain the following SCA Java annotations:
339	@Intent, @Qualifier. [JCA30006]
340 341	A Java interface referenced by the @interface attribute of an <interface.java></interface.java> element MUST NOT Field Code Changed contain any of the following SCA Java annotations:
342 343	@AllowsPassByReference, @ComponentName, @Constructor, @Context, @Destroy, @EagerInit, @Intent, @Property, @Qualifier, @Reference, @Scope, @Service. [JCA30007]
344	3.6 Compatibility of Java Interfaces
345 346 347 348 349	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. [JCA30009]

4 SCA Component Implementation Lifecycle

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

350

352

353

354

355 356

357

358

359

360

361 362

363 364

365

366

367 368

369

370

371

4.1 Overview of SCA Component Implementation Lifecycle

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

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

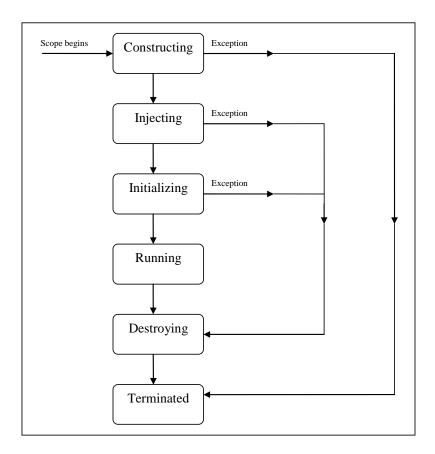
4.2 SCA Component Implementation Lifecycle State Diagram

The state diagram in

Figure 4.1 SCA - Component implementation lifecycle

<u>Figure 4-2</u> shows the lifecycle of an SCA component implementation. The sections that follow it describe each of the states that it contains.

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



372

373 374 375

376

377

378 379

380

381 382

383 384

385 386

387

388

389

390 391

Figure 4.1 SCA - Component implementation lifecycle

Figure 4-2: SCA - Component Implementation Lifecycle

4.2.1 Constructing State

The SCA Runtime MUST call a constructor of a component implementation. [JCA40001] When the constructor completes successfully, the SCA Runtime MUST transition the component implementation to the Injecting state. [JCA40002]

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

When the Constructing state, the SCA Runtime MUST transition the component implementation to the Injecting state. [JCA40003] If an exception is in the Injecting state, the SCA Runtime MUST transition the component implementation. [JCA40004]

4.2.2 Injecting State

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

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. [JCA40006] The order in which the references are injected is unspecified.

06 Feb 2010 Page 18 of 126

sca-javacaa-1.1-spec-<u>cd04</u> Copyright © OASIS® 2005, <u>2010.</u> All Rights Reserved.

392 The SCA Runtime MUST NOT invoke Service methods on the component implementation when the 393 component implementation is in the Injecting state. [JCA40007] The SCA Runtime MUST transition the component implementation to the Initializing state. [JCA40008] 394 395 The result of invoking operations on any injected references when the component implementation is in 396 the Injecting state is undefined. 397 When the injection of properties or references, the SCA Runtime MUST transition the component 398 implementation to the Destroying state. [JCA40009] If an exception is thrown whilst injecting properties or 399 references, the SCA Runtime MUST transition the component implementation to the Destroying state. [JCA40010] If a property or reference is unable to be injected, the SCA Runtime MUST transition the 400 component implementation to the Destroying state. [JCA40024] 401 402 4.2.3 Initializing State 403 When the component implementation enters the Initializing State, the SCA Runtime MUST call the method annotated with @Init on the component implementation, if present. [JCA40011] 404 405 The component implementation can invoke operations on any injected references when it is in the 406 Initializing state. However, depending on the order in which the component implementations are 407 initialized, the target of the injected reference might not be available since it has not yet been initialized. The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component 408 409 implementation instance is in the Initializing state. [JCA40012] 410 Once the method annotated with @Init completes successfully, the SCA Runtime MUST transition the component 411 implementation to the Running state. [JCA40013] If an exception is thrown whilst initializing, the SCA Runtime MUST transition the component implementation 412 413 to the Running state. [JCA40014] If an exception is thrown whilst initializing, the SCA Runtime MUST transition the component implementation to the Destroying state. [JCA40015] 414 4.2.4 Running State 415 When the component implementation scope ends, the SCA Runtime MUST transition the component implementation 416 417 to the Destroying state. [JCA40016] 418 The component implementation can invoke operations on any injected references when the component 419 implementation instance is in the Running state. 420 When a component implementation scope ends, the SCA Runtime MUST transition the component 421 implementation, if present. [JCA40017] 4.2.5 Destroying State 422 When a component implementation enters the Destroying state, the SCA Runtime MUST call the method 423 424 annotated with @Destroy on the component implementation, if present. [JCA40018] 425 The component implementation can invoke operations on any injected references when it is in the 426 Destroying state. However, depending on the order in which the component implementations are 427 destroyed, the target of the injected reference might no longer be available since it has been destroyed. 428 The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component 429 implementation instance is in the Destroying state. [JCA40019] 430 Once the method annotated with @Destroy completes successfully, the SCA Runtime MUST transition the

If an exception is thrown whilst destroying, the SCA Runtime MUST transition the component implementation to the Terminated state. [JCA40021] The SCA Runtime MUST transition the component implementation to

4.2.6 Terminated State

the Terminated state. [JCA40022]

431 432

433

434

435

436 The lifecycle of the SCA Component has ended.

component implementation to the Terminated state. [JCA40020]

5 Client API

439

442

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469 470

471

472 473

474 475

480

481 482

483

484

485

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

443 An SCA component can obtain a service reference either through injection or programmatically through 444 the *ComponentContext* API. Using reference injection is the recommended way to access a service, 445 since it results in code with minimal use of middleware APIs. The ComponentContext API is provided for 446 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: <#>ComponentContext¶ Non-SCA client code can use the ComponentContext API to perform operations against a component in an SCA domain. How client code obtains a reference to a ComponentContext is runtime specific. ¶ The following example demonstrates the use of the component Context API by non-SCA code: ¶ ComponentContext context = // obtained via host environment-specific means ¶ HelloService helloService = context.getService(HelloServ ice.class, "HelloService");¶ String result = helloService.hello("Hello

Formatted: Bullets and Numbering

World!");¶

```
486
487
488
                                           serviceURI);
                     String reply = helloService.sayHello("Mark");
                 489
490
491
492
493
494
       Snippet 5-1: Using the SCAClientFactory Interface
```

<u>For details about the SCAClientFactory interface and its related classes see the section "SCAClientFactory Class".</u>

495

496 497

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

6 Error Handling

- 499 Clients calling service methods can experience business exceptions and SCA runtime exceptions.
- Business exceptions are thrown by the implementation of the called service method, and are defined as 500 501 checked exceptions on the interface that types the service.
- 502 SCA runtime exceptions are raised by the SCA runtime and signal problems in management of
- component execution or problems interacting with remote services. The SCA runtime exceptions are 503
- 504 defined in the Java API section.

498

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

7 Asynchronous Programming

506 Asynchronous programming of a service is where a client invokes a service and carries on executing 507 without waiting for the service to execute. Typically, the invoked service executes at some later time. 508 Output from the invoked service, if any, is fed back to the client through a separate mechanism, since no 509 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 510 before the client continues. The SCA asynchronous programming model consists of: 511

- 512 support for non-blocking method calls
- 513

505

515 516

517

518

527 528

529 530

531

537

539

540

541

542

543

544

545

546

547

548

514 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.

519 A method with a void return type and which has no declared exceptions can be marked with a @OneWay

520 annotation. This means that the method is non-blocking and communication with the service provider can

521 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 522

Java client can use the JAX-WS asynchronous client API model that is described in the section "JAX-WS 523

Client Asynchronous API for a Synchronous Service". It is considered to be a best practice that service 524

525 designers define one-way methods as often as possible, in order to give the greatest degree of binding

flexibility to deployers. 526

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
- 532 a callback interface that is provided by the client

533 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 534 Model specification [ASSEMBLY]. 535

536 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 538 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: ny

sca-javacaa-1 1-spec-cd04 Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 24 of 126

```
549
550
          package somepackage;
551
           import org.oasisopen.sca.annotation.Callback;
552
           import org.oasisopen.sca.annotation.Remotable;
553
554
           @Remotable
555
          @Callback(QuotationCallback.class)
556
          public interface Quotation {h
557
               double requestQuotation(String productCode, int quantity);
558
559
560
          @Remotable
561
          public interface QuotationCallback {
562
               String getState();
563
               String getZipCode();
564
               String getCreditRating();
565
```

Snippet 7-1: Using a Bidirectional Interface

In <u>Snippet 7-1</u>, the <u>requestQuotation</u> 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

```
576
577
578
579
```

```
@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

Deleted: ¶
The code snippet below

<u>Snippet</u> 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.

```
public class ClientImpl implements ClientService, QuotationCallback {
   private QuotationService myService;

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

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 25 of 126

```
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

 Deleted: ¶
In this example

<u>Snippet</u> 7<u>-</u>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.

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]

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

sca-javacaa-1.1-spec-<u>cd04</u> Copyright © OASIS® 2005, <u>2010</u>. All Rights Reserved. 06 Feb 2010 Page 26 of 126

```
655
          @Callback
656
          protected MyServicelCallback callback1;
657
658
659
          protected MyService2Callback callback2;
660
```

Snippet 7-4: Multiple Bidirectional Interfaces in an Implementation

661 662

663

664 665 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.

667 668 669

666

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

670 671

672 673

674

675 676

677 678

679 680

681

<u>Snippet</u> 7-5 <u>comes</u> from a service implementation that uses the callback API:

Deleted: The following example fragments come

```
@Callback
protected ServiceReference<MyCallback> callback;
public void someMethod() {
  MyCallback myCallback = callback.getService();
  myCallback.receiveResult(theResult);
```

Snippet 7-5: Using the Callback API

682 683 684

685

686

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.

691

692 693

694 695

696 697 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();
                                                                                             Deleted: ¶
                                                                                             Deleted:
  myCallback.receiveResult(theResult);
```

698 699 700

Snippet 7-6: Using RequestContext to get a Callback

701 702 703

704

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

705 706 707

708

709 710

711 712

717 718 719

720 721 722

723 724

725 726

727

728 729 730

731 732

737

738 739 740

741 742 743

748

749 750

751 752

753

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 "asynclnyocation" 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> handler);
```

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
```

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

The main characteristics of the SCA asynchronous mapping are:

- there is a single method, with a name with the string "Async" appended to the operation name
- it has a void return type

758

759 760

761

762

763

764 765

766

767

768

769

770

771

772 773

774

775

776 777

778

779

780

781

782

783

784 785

786

787

788

789

790 791

792

793

794

795

796

797

798 799

800 801 802

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

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

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

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

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

The interface is annotated with the "asynclnvocation" intent.

- For each service operation in the WSDL, the Java interface contains an operation with
- a name which is the JAX-WS mapping of the WSDL operation name, with the suffix "Async" added
- a void return type
- a set of input parameter(s) which match the JAX-WS mapping of the input parameter(s) of the WSDL operation plus an additional last parameter which is a ResponseDispatch object typed by the JAX-WS Response Bean mapping of the output parameter(s) of the WSDL operation, where ResponseDispatch is the type defined in the SCA Java Common Annotations and APIs specification. [JCA60003]

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

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

003	i of the purposes of matering interfaces (when willing between a reference and a service, of when using
804	an implementation class by a component), an interface which has one or more methods which follow the
805	SCA asynchronous service pattern MUST be treated as if those methods are mapped as the equivalent
806	synchronous methods, as follows:
807	Asynchronous service methods are characterized by:
808	 void return type
809	 a method name with the suffix "Async"
810	 a last input parameter with a type of ResponseDispatch<x></x>
811	 annotation with the asynclnvocation intent
812	 possible annotation with the @AsyncFault annotation
813 814 815	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. [JCA60006]</x>

sca-javacaa-1.1-spec-<u>cd04</u>
Copyright © OASIS® 2005, <u>2010.</u> 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

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.

Deleted: the following:

```
861
862
              public static final String SCA_PREFIX =
863
                       "{http://docs.oasis-open.org/ns/opencsa/sca/200912}";
                                                                                                                    Deleted: 200903
864
              public static final String CONFIDENTIALITY =
                      SCA_PREFIX + "confidentiality";
865
              public static final String CONFIDENTIALITY_MESSAGE =
     CONFIDENTIALITY + ".message";
866
867
868
         Snippet 8-2: Example Intent Constants
869
870
       Notice that, by convention, qualified intents include the qualifier as part of the name of the constant,
871
       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
872
873
       section).
874
       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
875
                                                                                                                    Deleted: follows
876
       Snippet 8-3:
877
878
            @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE})
                                                                                                                    Deleted: ¶
879
         Snippet 8-3: Multiple Intnets in One Annotation
880
881
       The annotation in Snippet 8-3 attaches the intents "confidentiality.message" and "integrity.message".
                                                                                                                    Deleted: The following
882
       <u>Snippet</u> 8_4 is an example of a reference requiring support for confidentiality:
883
884
            package com.foo;
885
886
            import static org.oasisopen.sca.annotation.Confidentiality.*;
887
            import static org.oasisopen.sca.annotation.Reference;
888
            import static org.oasisopen.sca.annotation.Requires;
889
890
            public class Foo {
891
               @Requires(CONFIDENTIALITY)
892
               @Reference
893
               public void setBar(Bar bar) {
894
895
                                                                                                                    Deleted: }¶
896
897
         Snippet 8-4: Annotation a Reference
898
899
       Users can also choose to only use constants for the namespace part of the QName, so that they can add
900
       new intents without having to define new constants. In that case, the definition of Snippet 8-4 would
                                                                                                                    Deleted: this
901
       instead look like Snippet 8-5.
                                                                                                                    Deleted: this:
902
903
            package com.foo;
904
905
            import static org.oasisopen.sca.Constants.*;
906
            import static org.oasisopen.sca.annotation.Reference;
907
            import static org.oasisopen.sca.annotation.Requires;
908
909
            public class Foo {
910
               @Requires(SCA_PREFIX+"confidentiality")
```

06 Feb 2010

Page 32 of 126

911

912

@Reference

sca-iavacaa-1.1-spec-cd04

public void setBar(Bar bar) {

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

```
913
914
                                                                                                                       Deleted: }¶
915
916
         Snippet 8-5: Using Intent Constants and strings
917
       The formal syntax [EBNF-Syntax] for the @Requires annotation follows:
918
            '@Requires("' QualifiedIntent '"' (',"' QualifiedIntent '"')* ')'
919
920
       where
921
            QualifiedIntent ::= QName('.' Qualifier)*
922
            Qualifier ::= NCName
923
924
       See section @Requires for the formal definition of the @Requires annotation.
         8.2 Specific Intent Annotations
925
       In addition to the general intent annotation supplied by the @Requires annotation described in section
926
                                                                                                                       Deleted: above
       8.2, it is also possible to have Java annotations that correspond to specific policy intents. SCA provides a
927
928
       number of these specific intent annotations and it is also possible to create new specific intent
929
       annotations for any intent.
       The general form of these specific intent annotations is an annotation with a name derived from the name
930
931
       of the intent itself. If the intent is a qualified intent, qualifiers are supplied as an attribute to the annotation
932
       in the form of a string or an array of strings.
933
       For example, the SCA confidentiality intent described in the section on General Intent Annotations using
934
       the @Requires(CONFIDENTIALITY) annotation can also be specified with the @Confidentiality specific
                                                                                                                       Deleted: :
935
       intent annotation. The specific intent annotation for the "integrity" security intent is shown in Snippet 8-6.
936
937
            @Integrity
938
         Snippet 8-6: Example Specific Intent Annotation
939
                                                                                                                       Deleted: :
940
       An example of a qualified specific intent for the "authentication" intent is shown in Snippet 8-7.
941
942
            @Authentication( {"message", "transport"} )
943
         Snippet 8-7: Example Qualified Specific Intent Annotation
944
       This annotation attaches the pair of qualified intents: "authentication.message" and
945
946
        "authentication.transport" (the sca: namespace is assumed in this both of these cases -
                                                                                                                       Deleted: 200903
       "http://docs.oasis-open.org/ns/opencsa/sca/200912").
947
                                                                                                                       Deleted: :
       The general form of specific intent annotations is shown in Snippet 8-8
948
949
950
            '@' Intent ('(' qualifiers ')')?
951
       where Intent is an NCName that denotes a particular type of intent.
952
                               ::= NCName
            Intent.
                               ::= '"' qualifier '"' (',"' qualifier '"')*
953
            qualifiers
954
            qualifier::= NCName ('.' qualifier)?
955
         Snippet 8-8: Specific Intent Annotation Format
```

06 Feb 2010

Page 33 of 126

sca-iavacaa-1.1-spec-cd04

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:

Deleted: for example:

Deleted: .

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

Snippet 8-9: Defining a Specific Intent Annotation

968 969 970

971

972

973

974 975

976

977

978

956 957

958

959

960

961

962

963 964

965 966

967

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

Deleted: :

Deleted:

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

Snippet 8-10: Multiple Qualifiers in an Annotation'

979 980 981

982

984

985

986 987

990

992

993

994

995

996

997

998

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

983 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:

- 988 Java class
- 989 Java interface
 - Method
- 991 Field
 - Constructor parameter
 - 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.

[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.

```
A annotations (general or specific) are applied to the same Java element, the SCA runtime MUST
 999
                                                                                                                       Field Code Changed
1000
        NOT instantiate such an implementation class. [JCA70003]
                                                                                                                       Deleted: follows
1001
        An example of multiple policy annotations being used together is shown in Snippet 8-11:
1002
1003
             @Authentication
1004
             @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE})
1005
          Snippet 8-11: Multiple Policy Annotations
1006
1007
        In this case, the effective intents are "authentication", "confidentiality.message" and "integrity.message".
1008
        If a constructor of an implementation class is annotated with @Constructor and the constructor has parameters, each
1009
        of these parameters MUST have either a @Property annotation or a @Reference annotation. [JCA70004] This
        merging process does not remove or change any intents that are applied to the interface.
1010
          8.3.1 Intent Annotation Examples
1011
1012
        The following examples show how the rules defined in section 8.3 are applied.
                                                                                                                       Deleted: Example 8.1
1013
        Snippet 8-12 shows how intents on references are merged. In this example, the intents for myRef are
1014
         "authentication" and "confidentiality.message".
1015
1016
             @Authentication
1017
             @Requires(CONFIDENTIALITY)
1018
             @Confidentiality("message")
1019
             @Reference
1020
             protected MyService myRef;
                                                                                                                       Deleted:
1021
           Snippet 8-12: Merging Intents on References
                                                                                                                       Deleted: Example 8.1
1022
                                                                                                                       Deleted: intents
                                                                                                                       Deleted: references.
1023
        Snippet 8-13 shows that mutually exclusive intents cannot be applied to the same Java element. In this
1024
        example, the Java code is in error because of contradictory mutually exclusive intents
                                                                                                                       Deleted: Example 8.2
1025
         "managedTransaction" and "noManagedTransaction".
1026
1027
             @Requires({SCA_PREFIX+"managedTransaction",
1028
                          SCA PREFIX+"noManagedTransaction" })
1029
             @Reference
1030
             protected MyService myRef;
                                                                                                                       Deleted:
1031
          Snippet 8-13: Mutually Exclusive Intents
                                                                                                                       Deleted: Example 8.2.
1032
                                                                                                                       Deleted: exclusive intents.
                                                                                                                       Deleted: Example 8.3
1033
        Snippet 8-14 shows that intents can be applied to Java service interfaces and their methods. In this
1034
        example, the effective intents for MyService.mymethod() are "authentication" and "confidentiality".
1035
1036
             @Authentication
1037
             public interface MyService {
1038
                  @Confidentiality
1039
                  public void mymethod();
1040
1041
             @Service(MyService.class)
1042
             public class MyServiceImpl
                                                                                                                       Deleted: Example 8.3.
1043
                  public void mymethod() {...}
                                                                                                                       Deleted: interfaces, interface
1044
                                                                                                                       methods
1045
           Snippet 8-14: Intents on Java Interfaces, Interface Methods, and Java Classes
                                                                                                                       Deleted: classes
1046
                                                                                                                       Deleted: Example 8.4
        sca-iavacaa-1.1-spec-cd04
                                                                                                     06 Feb 2010
```

Page 35 of 126

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

```
1049
         "managedTransaction".
1050
1051
             @Authentication
1052
             public interface MyService {
1053
                  @Confidentiality
1054
                  public void mymethod();
1055
1056
             @Service(MyService.class)
1057
             @Requires(SCA_PREFIX+"managedTransaction")
1058
             public class MyServiceImpl
1059
                  public void mymethod() {...}
1060
                                                                                                                       Deleted: Example 8.4.
1061
           Snippet 8-15: Intents on Java Service Implementation Classes
                                                                                                                       Deleted: service implementation
                                                                                                                       classes
1062
                                                                                                                       Deleted: Example 8.5
1063
        Snippet 8-16 shows that intents can be applied to Java reference interfaces and their methods, and also
1064
        to Java references. In this example, the effective intents for the method ( ) of the reference
1065
        myRef are "authentication", "integrity", and "confidentiality".
1066
1067
             @Authentication
1068
             public interface MyRefInt {
1069
                  @Integrity
1070
                  public void mymethod();
1071
1072
             @Service(MyService.class)
1073
             public class MyServiceImpl {
1074
                  @Confidentiality
1075
                  @Reference
1076
                  protected MyRefInt myRef;
1077
                                                                                                                       Deleted: Example 8.5.
1078
          <u>Snippet</u> 8-16: Intents on Java <u>References</u> and their <u>Interfaces</u> and <u>Methods</u>
                                                                                                                       Deleted: references
1079
                                                                                                                       Deleted: interfaces
1080
        Snippet 8-17 shows that intents cannot be applied to methods of Java implementation classes. In this
                                                                                                                       Deleted: methods.
1081
        example, the Java code is in error because of the @Authentication intent annotation on the
                                                                                                                       Deleted: Example 8.6
1082
        implementation method MyServiceImpl.mymethod().
1083
1084
             public interface MyService {
1085
                  public void mymethod();
1086
1087
             @Service(MyService.class)
1088
             public class MyServiceImpl {
1089
                  @Authentication
1090
                  public void mymethod() {...}
1091
                                                                                                                       Deleted: Example 8.6.
1092
          Snippet 8-17: Intent on Implementation Method
                                                                                                                       Deleted: implementation method.
1093
        Snippet 8-18 shows one effect of applying the SCA Policy Framework rules for merging intents within a
                                                                                                                       Deleted: Example 8.7
1094
        structural hierarchy to Java service interfaces and their methods. In this example a qualified intent
1095
        overrides an unqualified intent, so the effective intent for MyService.mymethod() is
1096
         "confidentiality.message".
1097
1098
             @Confidentiality("message")
1099
             public interface MyService {
        sca-iavacaa-1.1-spec-cd04
                                                                                                     06 Feb 2010
        Copyright © OASIS® 2005, 2010. All Rights Reserved.
                                                                                                   Page 36 of 126
```

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

1047

1048

```
1100
                   @Confidentiality
1101
                   public void mymethod();
1102
                                                                                                                            Deleted:
1103
           <u> Snippet 8-18:</u> Merging <u>Qualified</u> and <u>Unqualified Intents</u> on Java <u>Interfaces</u> and <u>Methods</u>
                                                                                                                            Deleted: Example 8.7.
                                                                                                                            Deleted: qualified
1104
                                                                                                                            Deleted: unqualified intents
1105
         Snippet 8-19 shows another effect of applying the SCA Policy Framework rules for merging intents within
1106
         a structural hierarchy to Java service interfaces and their methods. In this example a lower-level intent
                                                                                                                            Deleted: interfaces
1107
         causes a mutually exclusive higher-level intent to be ignored, so the effective intent for mymethod1() is
                                                                                                                            Deleted: methods.
1108
         "managedTransaction" and the effective intent for mymethod2() is "noManagedTransaction".
                                                                                                                            Deleted: Example 8.8
1109
1110
             @Requires(SCA_PREFIX+"managedTransaction")
1111
              public interface MyService {
1112
                  public void mymethod1();
                   @Requires(SCA_PREFIX+"noManagedTransaction")
1113
1114
                   public void mymethod2();
```

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

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

8.3.2 Inheritance and Annotation

1115

1116

1117

1118

1119 1120

1121

1122

1123 1124

1125

1126

1127

1128

1129

1130 1131

1132

1133

1134 1135 1136

1137

1138

1139 1140

1141

1142

1143 1144

1145

1146

1147

1148 1149

1150 1151

Snippet 8-20 shows the inheritance relations of intents on classes, operations, and super classes.

Deleted: The following example

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

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

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 1152 1153 @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")

1155 1156 1157

1158

1159

1160

1161

1162 1163

1164 1165

1166

1167

1168 1169

1170

1171

1172

1173

1174 1175 1176

1177

1178

1181 1182

1183

1184

1185

1186 1187 1188

1154

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

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

"Table 8-1: Declarative Intents Equivalent to Annotated Intents in Snippet 8-20

Deleted: ¶ Table 8.1. Declarative intents equivalent to annotated intents in Example 8.9.¶

8.4 Relationship of Declarative and Annotated Intents

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

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

8.5 Policy Set Annotations

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

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:

Snippet 8-21: PolicySet Annotation Format

1179 As for intents, PolicySet names are QNames - in the form of "{Namespace-URI}localPart". 1180

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

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

Snippet 8-22: Use of @PolicySets

sca-iavacaa-1.1-spec-cd04 Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 38 of 126 1189 1190

1200

1201

1207

1208

1209

1210 1211

1212

1213

1214

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232 1233 1234

1235

In this case, the Policy Sets WS_Encryption_Policy and WS_Authentication_Policy are applied, both

using the namespace defined for the constant MY_NS.

PolicySets need to satisfy intents expressed for the implementation when both are present, according to

the rules defined in the Policy Framework specification [POLICY].
 The SCA Policy Set annotation can be applied to the following Java elements:

- 1195 Java class
- 1196 Java interface
- 1197 Method
- 1198 Field
- 1199 Constructor parameter
 - A method annotated with @Destroy MAY have any access modifier and MUST have a void return type and no arguments.

1202 [JCA70005]

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

If 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. [JCA70006] This merging process does not remove or change any policy sets that are applied to the interface.

8.6 Security Policy Annotations

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

- 1215 @Authentication
- 1216 @Authorization
- 1217 @Confidentiality
- 1218 @Integrity
- 1219 @MutualAuthentication

1220 The @Authentication, @Confidentiality, and @Integrity intents have the same pair of Qualifiers:

- 1221 message
- 1222 transport

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

<u>Snippet</u> 8-23 shows an example of applying <u>security intents to</u> 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
```

sca-javacaa-1.1-spec-<u>cd04</u> Copyright © OASIS® 2005, <u>2010.</u> All Rights Reserved. Deleted: SCA's

Deleted: specification

Deleted: . ¶

<#>Security Interaction Policy¶
The following interaction policy
Intents and qualifiers are defined
for Security Policy, which apply
to the operation of services and
references of an implementation:
<#>@Integrity¶

Deleted: ¶
All three of these

Deleted: @Authentication, @Confidentality and @Integrity

Deleted: sections @ Authentication, @ Confidentiality and @ Integrity.

Deleted: The following example

Deleted: an intent to

Deleted:

06 Feb 2010 Page 39 of 126

```
1236
           public interface ClientService {
1237
1238
              public void clientMethod();
1239
1240
           // Implementation class for ClientService
1241
           package services.client;
1242
1243
           import services.hello.HelloService;
1244
           import org.oasisopen.sca.annotation.*;
1245
1246
           @Service(ClientService.class)
1247
           public class ClientServiceImpl implements ClientService {
1248
1249
              private HelloService helloService;
1250
1251
              @Reference(name="helloService", required=true)
1252
              @Integrity("message")
1253
              @Authentication("message")
1254
              public void setHelloService(HelloService service) {
1255
                     helloService = service;
1256
              }
1257
1258
              public void clientMethod() {
1259
                     String result = helloService.hello("Hello World!");
1260
1261
              }
1262
```

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

Deleted: ¶ Example 8

Example 8.10. Usage of annotated 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
- 1274 <u>local</u>

1263

1264 1265

1266

1267

1268

1269

1270

1271 1272

1273

1275

1276

1277

1278 1279

1280

1281

1286

1287 1288

1289

1290

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

<u>Snippet 8-24 shows an example of applying a transaction intent to a component implementation, where</u> the component implementation requires a global transaction.

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

// Implementation class for HelloService
package services.hello.impl;

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

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 40 of 126

```
1291
1292
1293
1294
1295
1296
1297
1298
                 @Service(HelloService.class)
                 @ManagedTransaction("global")
public class HelloServiceImpl implements HelloService {
                  public void someMethod() {
1299
```

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

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

9 Java API

1300 1301

1302

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

9.1 Component Context

```
Deleted: The following Java code
       Figure 9-1 defines the ComponentContext interface:
1303
1304
1305
            package org.oasisopen.sca;
           import java.util.Collection;
public interface ComponentContext {
1306
                                                                                                           Deleted:
1307
1308
1309
                String getURI();
1310
1311
            Deleted:
1312
1313
               <B> ServiceReference<B> getServiceReference(_Class<B> businessInterface,
1314
                                                               String referenceName);
                                                                                                           Deleted:
1315
               <B> Collection<B> getServices( Class<B> businessInterface,
1316
                                                String referenceName);
1317
1318
               <B> Collection<ServiceReference<B>> getServiceReferences(
1319
                                                           Class<B> businessInterface,
                                                                                                           Deleted: ¶
1320
                                                           String referenceName);
1321
1322
               <B> ServiceReference<B> createSelfReference(Class<B> businessInterface);
                                                                                                           Deleted: ¶
1323
1324
               <B> ServiceReference<B> createSelfReference( Class<B> businessInterface,
1325
1326
                                                              String serviceName);
                                                                                                           Deleted:
1327
               <B> B getProperty(Class<B> type, String propertyName);
1328
1329
               RequestContext getRequestContext();
1330
1331
               <B> ServiceReference<B> cast(B target) throws IllegalArgumentException;
1332
1333
1334
           Figure 9-1: ComponentContext Interface
1335
                                                                                                           Deleted: () - returns
1336
       getURL() method:
                                                                                                           Deleted: domain
1337
       Returns the absolute URI of the component within the SCA Domain.
1338
       Returns:
1339
           String which contains the absolute URI of the component in the SCA Domain
1340
           The ComponentContext.getURI method MUST return the absolute URI of the component in the SCA
           Domain. [JCA80008]
1341
1342
       Parameters:
1343
           none
1344
       Exceptions:
1345
           none
1346
1347
                  getService(Class<B> businessInterface, String referenceName) - Returns a proxy for
1348
                  the reference defined by the current component. The getService() method takes as its
       sca-iavacaa-1.1-spec-cd04
                                                                                           06 Feb 2010
       Copyright © OASIS® 2005, 2010. All Rights Reserved.
```

Page 42 of 126

1349 1350 1351	input arguments the Java type used to represent the target service on the client and the name of the service reference. It returns an object providing access to the service. The returned object implements the Java interface the service is typed with. [JCA80001]
1352	<u>getServiceReference((</u> Class businessInterface, String referenceName,) <u>method:</u>
1353 1354	Returns a typed service proxy object for a reference defined by the current component, where the reference has multiplicity 01 or 11.
1355	Returns:
1356 1357	B which is a proxy object for the reference, which implements the interface B contained in the businessInterface parameter.
1358 1359 1360 1361	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]
1362 1363 1364	The ComponentContext.getService method MUST return null if the multiplicity of the reference named by the referenceName parameter is 01 and the reference has no target service configured. [JCA80010]
1365	Parameters:
1366	Class businessInterface - the Java interface for the service reference
1367	String referenceName - the name of the service reference
1368	Exceptions:
1369	• <u>[JCA80001]</u>
1370 1371 1372	 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]
1373 1374 1375	• 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]
1376 1377	getServiceReference (Class businessInterface, String referenceName) method:
1378 1379	Returns a ServiceReference object for a reference defined by the current component where the reference has multiplicity 01 or 11.
1380	Returns:
1381 1382 1383 1384 1385 1386 1387 1388 1389	• ServiceReference 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 01 and the reference has no target service configured. [JCA80007]
1390	Parameters:
1391	Class businessInterface - the Java interface for the service reference
1392	String referenceName - the name of the service reference
1393	Exceptions:
1394 1395	 The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if the reference named by the referenceName parameter has multiplicity greater than one. [JCA80004]

Deleted: . This method MUST throw an IllegalArgumentException if the reference has multiplicity greater than one.

Deleted:)-

1396 The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if 1397 the reference named by the referenceName parameter does not have an interface of the type defined 1398 by the businessInterface parameter, [JCA80005] 1399 The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if 1400 he component does not have a reference with the name provided in the referenceName parameter. [JCA80006] 1401 1402 1403 getServices(Class businessInterface, String referenceName).method: 1404 Returns a list of typed service proxies for a reference defined by the current component, where the 1405 reference has multiplicity 0..n or 1..n.

Deleted: business interface type and

Deleted: -

a reference name.

Returns:

1406

1407

1408

1409

1410

1411

14121413

1414

1415

1417

1418

1419

1420 1421

1422

1423 1424

1425

1426

1427

1428

1429 1430

1431

1433

1434

1435 1436

1437

1438

1439

1440

1441 1442

1443

• Collection 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]

1416 Parameters:

- Class 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 businessInterface, String referenceName) method:

Deleted: -

reference name.

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

1432 <u>Returns:</u>

• Collection<ServiceReference> 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]

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010

Page 44 of 126

Deleted: service references for a business interface type and a

1444 Parameters: 1445 Class businessInterface - the Java interface for the service reference 1446 String referenceName - the name of the service reference 1447 The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if 1448 1449 the reference identified by the referenceName parameter has multiplicity of 0..1 or 1..1. [JCA80021] 1450 The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter. 1451 [JCA80022] 1452 1453 The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if he service reference with the name supplied in the referenceName does not have an interface 1454 compatible with the interface supplied in the businessInterface parameter. [JCA80023] 1455 1456 createSelfReference(Class businessInterface) method: 1457 1458 Returns a ServiceReference object that can be used to invoke this component over the designated 1459 service. 1460 Returns: 1461 ServiceReference which is a ServiceReference object for the service of this component which 1462 has the supplied business interface. If the component has multiple services with the same business 1463 interface the SCA runtime can return a ServiceReference for any one of them. The ComponentContext.createSelfReference method MUST return a ServiceReference object typed 1464 by the interface defined by the businessInterface parameter for one of the services of the invoking 1465 component which has the interface defined by the businessInterface parameter. [JCA80024] 1466 1467 Parameters: 1468 • Class businessInterface - the Java interface for the service 1469 **Exceptions:** The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the component does not have a service which implements the interface identified by the 1470 1471 businessInterface parameter. [JCA80025] 1472 1473 1474 createSelfReference(Class businessInterface, String serviceName)_method: 1475 Returns a ServiceReference that can be used to invoke this component over the designated service. The 1476 serviceName parameter explicitly declares the service name to invoke 1477 1478 ServiceReference which is a ServiceReference proxy object for the reference, which implements 1479 the interface contained in the businessInterface parameter. 1480 The ComponentContext.createSelfReference method MUST return a ServiceReference object typed by the interface defined by the businessInterface parameter for the service identified by the 1481 serviceName of the invoking component and which has the interface defined by the businessInterface 1482 1483 parameter. [JCA80026] 1484 Parameters: 1485 Class businessInterface - the Java interface for the service reference 1486 String serviceName - the name of the service reference 1487 Exceptions: 1488 The ComponentContext.createSelfReference method MUST throw an IllegalArgumentException if the 1489 component does not have a service with the name identified by the serviceName parameter.

1490

[JCA80027] sca-javacaa-1.1-spec-<u>cd04</u>

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

06 Feb 2010 Page 45 of 126 Deleted: - Returns a

ServiceReference

Deleted: -

1491 1492 1493	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]	
1494 1495	getProperty (Class type, String propertyName) method:	Deleted: -
1496	Returns the value of an SCA property defined by this component.	
1497	Returns:	
1498 1499 1500 1501 1502 1503		
1504	Parameters:	
1505 1506	Class type - the Java class of the property (Object mapped type for primitive Java types - e.g. Integer if the type is int)	
1507	String propertyName - the name of the property	
508	Exceptions:	
1509 1510 1511	 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] 	
1512 1513	 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 	
1514	which is compatible with the supplied type parameter. [JCA80031]	
I515 I516	getRequestContext()_method:	Deleted: -
1517	Returns the RequestContext for the current SCA service request.	Deleted: context
1518	Returns:	Deleted: , or
1519 1520	 <u>RequestContext</u> which is the <u>RequestContext object for the current SCA service invocation.</u> null if there is no current request or if the context is unavailable. 	
521 522	When marked for eager initialization with an @EagerInit annotation, the composite scoped instance MUST be created when its containing component is started. [JCA80002]	
1523	Parameters:	
1524	• <u>none</u>	
1525 1526	Exceptions:	
1527	• none	
1528	cast(B target) <u>method:</u>	Deleted: -
1529	Casts a type-safe reference to a ServiceReference	
1530	Returns:	
1531 1532	ServiceReference > which is a ServiceReference object which implements the same business interface B as a reference proxy object	
1533 1534 1535	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]	
1536	Parameters:	
. 500	sca-javacaa-1.1-spec- <u>cd04</u> 06 Feb 2010	
	Copyright © OASIS® 2005, 2010. All Rights Reserved. Page 46 of 126	

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

Exceptions:

}

 The ComponentContext.cast method MUST throw an IllegalArgumentException if the supplied target parameter is not an SCA reference proxy object. [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(...)**.

<u>Snippet 9-1</u> shows an example of component context usage in a Java class using the @Context annotation.

```
Deleted: The following
```

Deleted:

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

<u>Figure 9-2</u> shows the *RequestContext* interface:

```
Deleted: The following
```

Deleted: ¶

```
package org.oasisopen.sca;
import javax.security.auth.Subject;
public interface RequestContext {
   Subject getSecuritySubject();
   String getServiceName();
        <CB> ServiceReference<CB> getCallbackReference();
        <CB> CB getCallback();
        <B> ServiceReference<B> getServiceReference();
```

The RequestContext interface has the following methods:

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

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010

Page 47 of 126

1588 1589 1590	 getCallback() – Returns a proxy for the callback as specified by the caller. Similar to the getCallbackReference() method, this method returns null when called for a service reques whose interface is not bidirectional or when called for a callback request.
1591 1592 1593	 getServiceReference() – 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]
1594	9.3 <u>ServiceReference</u>
1595	<u>}</u>
1596	Figure 9-2: RequestContext Interface
1597	
1598	getSecuritySubject () method:
1599	Returns the JAAS Subject of the current request (see the JAAS Reference Guide [JAAS] for details of
1600	JAAS).
1601	Returns:
1602	 javax.security.auth.Subject object which is the JAAS subject for the request.
1603	null if there is no subject for the request.
1604 1605	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
1606	service request or callback request. [JCA80034]
1607	Parameters:
1608	• none
1609	Exceptions:
1610	• none
1611	
1612	getServiceName () method:
1613	Returns the name of the service on the Java implementation the request came in on.
1614	Returns:
1615 1616	• 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.
1617	The RequestContext.getServiceName method MUST return the name of the service for which an
1618 1619	operation is being processed, or null if invoked from a thread that is not processing a service operation or a callback operation. [JCA80035]
1620	Parameters:
1621	• none
1622	Exceptions:
1623	• none
1624	
1625	getCallbackReference () method:
1626 1627	Returns a service reference proxy for the callback for the invoked service operation, as specified by the service client.
1628	Returns:
1629 1630	• 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.</cb>

1631 1632	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.
1633	null if the method is invoked from a thread that is not processing a service operation.
1634 1635 1636 1637	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]
1638	Parameters:
1639	• <u>none</u>
1640	Exceptions:
1641	• <u>none</u>
1642	
1643	getCallback () method:
1644	Returns a proxy for the callback for the invoked service as specified by the service client.
1645	Returns:
1646 1647	<u>CB</u> proxy object for the callback for the invoked service as supplied by the service client. It is typed with the callback interface.
1648 1649	<u>null</u> 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.
1650	null if the method is invoked from a thread that is not processing a service operation.
1651 1652 1653 1654	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]
1655	Parameters:
1656	• <u>none</u>
1657	Exceptions:
1658	• <u>none</u>
1659	
1660	getServiceReference () method:
1661	Returns a ServiceReference object for the service that was invoked.
1662	Returns:
1663 1664	• <u>ServiceReference</u> which is a service reference for the invoked service. It is typed with the interface of the service.
1665 1666	null if the method is invoked from a thread that is not processing a service operation or a callback operation.
1667 1668	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]
1669 1670	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]
1671 1672	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]
1673	Parameters:
1674	• <u>none</u>

1675 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.

9.4 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.

Deleted: following Java code

Deleted: interface has the following

methods:

Deleted: () -

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

1684 1685

1695

1696

1697 1698

1699 1700

1701

1702

1703

1704

1705

1706

1708

1676

1677

1678 1679

1680 1681

1682

1683

```
1686
           package org.oasisopen.sca;
1687
1688
           public interface ServiceReference<B> extends java.io.Serializable {
1689
                                                                                                       Deleted:
1690
1691
                 B getService();
1692
                  Class<B> getBusinessInterface();
                                                                                                       Deleted: }¶
1693
                                                                                                       The
1694
         <u>Figure</u> 9-3: ServiceReference <u>Interface</u>
```

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:

• 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:

1707 • <u>none</u>

Exceptions:

• none

1709 1710 1711

1712 1713 1714

1715

1716

getBusinessInterface () method:

Returns the Java class for the business interface associated with this <u>ServiceReference</u>.

Deleted: () –

Deleted: reference

Returns:

• Class 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]

1717 Parameters:

1718 • <u>none</u>

1719 Exceptions:

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 50 of 126 1720 • <u>none</u>

9.5 ResponseDispatch interface

The ResponseDispatch interface is shown in Figure 9-4:

17211722

1723

```
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

173117321733

1734

1735 1736

1739 1740

1742 1743

1730

sendResponse (T response) method:

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

1737 <u>Returns:</u>

1738 • <u>void</u>

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

1741 Parameters:

• <u>T - an instance of the response message returned by the service operation</u>

Exceptions:

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

1745 1746 1747

1748

1749 1750

1752

1753

1754

1755

1756

1744

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.

1751 Returns:

• <u>void</u>

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

Parameters:

• <u>e - an instance of an exception returned by the service operation</u>

1757 <u>Exceptions:</u>

• 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:

1762 Obtains the context object for the ResponseDispatch method

1763 <u>Returns:</u>

```
1764
            Map<String, object> which is the context object for the ResponseDispatch object.
1765
            The invoker can update the context object with appropriate context information, prior to invoking
1766
            either the sendResponse method or the sendFault method
1767
        Parameters:
1768
           none
1769
        Exceptions:
1770
           none
          9.6 ServiceRuntimeException
1771
                                                                                                                  Deleted: The following snippet
        Figure 9-5 shows the ServiceRuntimeException.
1772
1773
1774
            package org.oasisopen.sca;
1775
1776
            public class ServiceRuntimeException extends RuntimeException {
1777
1778
1779
          Figure 9-5: ServiceRuntimeException
1780
1781
        This exception signals problems in the management of SCA component execution.
          9.7 ServiceUnavailableException
1782
                                                                                                                  Deleted: The following snippet
1783
        Figure 9-6 shows the ServiceUnavailableException.
1784
1785
            package org.oasisopen.sca;
1786
1787
            public class ServiceUnavailableException extends ServiceRuntimeException {
1788
1789
                                                                                                                  Deleted: }¶
1790
          Figure 9-6: ServiceUnavailableException
1791
1792
        This exception signals problems in the interaction with remote services. These are exceptions that can
1793
        be transient, so retrying is appropriate. Any exception that is a ServiceRuntimeException that is not a
1794
        ServiceUnavailableException is unlikely to be resolved by retrying the operation, since it most likely
        requires human intervention
1795
          9.8 InvalidServiceException
1796
                                                                                                                  Deleted: The following snippet
        Figure 9-7 shows the InvalidServiceException.
1797
1798
1799
            package org.oasisopen.sca;
1800
1801
            public class InvalidServiceException extends ServiceRuntimeException {
1802
1803
                                                                                                                  Deleted: }¶
1804
          Figure 9-7: InvalidServiceException
1805
```

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.

9.9 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:

```
Deleted: =

Deleted: =

Deleted: 200903
```

Deleted: }

Deleted: The following snippet

```
package org.oasisopen.sca;
public interface Constants {
    String SCA_NS = "http://docs.oasis-open.org/ns/opencsa/sca/200912";
    String SCA_PREFIX = "{"+SCA_NS+"}";
    String SERVERAUTHENTICATION = SCA_PREFIX + "serverAuthentication";
    String CLIENTAUTHENTICATION = SCA_PREFIX + "clientAuthentication";
    String ATLEASTONCE = SCA_PREFIX + "atLeastOnce";
String ATMOSTONCE = SCA_PREFIX + "atMostOnce";
    String EXACTLYONCE = SCA_PREFIX + "exactlyOnce";
    String ORDERED = SCA_PREFIX + "ordered";
    String TRANSACTEDONEWAY = SCA_PREFIX + "transactedOneWay";
    String IMMEDIATEONEWAY = SCA_PREFIX + "immediateOneWay";
    String PROPAGATESTRANSACTION = SCA_PREFIX + "propagatesTransaction";
    String SUSPENDSTRANSACTION = SCA_PREFIX + "suspendsTransaction";
    String ASYNCINVOCATION = SCA_PREFIX + "asyncInvocation";
    String SOAP = SCA_PREFIX + "SOAP";
    String JMS = SCA_PREFIX + "JMS";
                                       "noListener";
    String NOLISTENER = SCA_PREFIX +
    String EJB = SCA_PREFIX + "EJB";
```

Figure 9-8: Constants Interface

9.10 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:

```
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;
```

sca-javacaa-1.1-spec-cd04 Copyright © OASIS® 2005, 2010. All Rights Reserved. 06 Feb 2010 Page 53 of 126

```
1860
            public abstract class SCAClientFactory {
1861
1862
                protected static SCAClientFactoryFinder factoryFinder;
1863
1864
                private URI domainURI;
1865
1866
                private SCAClientFactory() {
1867
1868
1869
                protected SCAClientFactory(URI domainURI)
1870
                      throws NoSuchDomainException {
1871
                    this.domainURI = domainURI;
1872
1873
1874
                protected URI getDomainURI() {
1875
                    return domainURI;
1876
1877
1878
                public static SCAClientFactory newInstance( URI domainURI )
1879
                    throws NoSuchDomainException {
return newInstance(null, null, domainURI);
1880
1881
1882
1883
                public static SCAClientFactory newInstance(Properties properties,
1884
                                                                               URI domainURI)
1885
                      throws NoSuchDomainException {
1886
                    return newInstance(properties, null, domainURI);
1887
1888
1889
                public static SCAClientFactory newInstance(ClassLoader classLoader,
1890
                      throws NoSuchDomainException {
1891
1892
                    return newInstance(null, classLoader, domainURI);
1893
1894
1895
                public static SCAClientFactory newInstance(Properties properties,
                                                           ClassLoader classLoader,
1896
1897
                                                           URI domainURI)
1898
                      throws NoSuchDomainException
1899
                    final SCAClientFactoryFinder finder =
1900
                         factoryFinder != null ? factoryFinder :
    new SCAClientFactoryFinderImpl();
1901
1902
                    final SCAClientFactory factory
1903
                           finder.find(properties, classLoader, domainURI);
1904
                    return factory;
1905
1906
1907
                public abstract <T> T getService(Class<T> interfaze, String serviceURI)
1908
                    throws NoSuchServiceException, NoSuchDomainException;
1909
1910
```

Figure 9-9: SCAClientFactory Class

newInstance (URI domainURI) method:

Obtains a object implementing the SCAClientFactory class.

Returns:

1911 1912

1913

1914

1915

1916

1917 1918 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:

sca-iavacaa-1.1-spec-cd04 Copyright © OASIS® 2005, 2010. All Rights Reserved.

Exceptions: The SCAClientFactory.newInstance(URI) method MUST throw domainURI parameter does not identify a valid SCA Domain. [JC newInstance(Properties properties, URI domainURI) method: Obtains a object implementing the SCAClientFactory class, using a sReturns: object which implements the SCAClientFactory class. The SCAClientFactory.newInstance(Properties, URI) method Mimplements the SCAClientFactory class for the SCA Domain identification of the SCAClientFactory class for the SCA Domain identification of the SCAClientFactory class. properties - a set of Properties that can be used when creating the SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(Properties, URI) method MinoSuchDomainException if the domainURI parameter does not in JCA80045] mewInstance(ClassIoader classLoader, URI domainURI) method. Obtains a object implementing the SCAClientFactory class using a signerum of the SCAClientFactory class. by object which implements the SCAClientFactory class. classLoader - a ClassLoader to use when creating the object with SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in JCA80047] mewInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in JCA80047]	
domainURI parameter does not identify a valid SCA Domain. [JC newInstance(Properties properties, URI domainURI) method: Obtains a object implementing the SCAClientFactory class, using a sexturns: • object which implements the SCAClientFactory class The SCAClientFactory.newInstance(Properties, URI) method Mimplements the SCAClientFactory class for the SCA Domain ider [JCA80044] Parameters: • properties - a set of Properties that can be used when creating the SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Properties, URI) method MinoSuchDomainException if the domainURI parameter does not in JCA80045] mewInstance(ClassIoader classLoader, URI domainURI) method MinoSuchDomainException if the SCAClientFactory class using a sexturns: • object which implements the SCAClientFactory class The SCAClientFactory.newInstance(ClassIoader, URI) method implements the SCAClientFactory class for the SCA Domain ider [JCA80046] Parameters: • classLoader - a ClassLoader to use when creating the object with SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(ClassIoader, URI) method NosuchDomainException if the domainURI parameter does not in JCA80047] newInstance(Properties properties, ClassIoader classLoader, URI) method NosuchDomainException if the domainURI parameter does not in JCA80047]	
newInstance(Properties properties, URI domainURI) method: Obtains a object implementing the SCAClientFactory class, using a sReturns: • object which implements the SCAClientFactory class The SCAClientFactory.newInstance(Properties, URI) method Mimplements the SCAClientFactory class for the SCA Domain ider [JCA80044] Parameters: • properties - a set of Properties that can be used when creating the SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Properties, URI) method MinosuchDomainException if the domainURI parameter does not in Jucasous Jucaso	
Obtains a object implementing the SCAClientFactory class, using a sReturns: • object which implements the SCAClientFactory class The SCAClientFactory.newInstance(Properties. URI) method Mimplements the SCAClientFactory class for the SCA Domain idea [JCA80044] Parameters: • properties - a set of Properties that can be used when creating the SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Properties, URI) method MinoSuchDomainException if the domainURI parameter does not in JCA80045] mewInstance(ClassIoader classLoader, URI domainURI) method. Obtains a object implementing the SCAClientFactory class using a signer. • object which implements the SCAClientFactory class. • object which implements the SCAClientFactory class. • classLoader - a ClassLoader to use when creating the object with SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in JCA80047] mewInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in JCA80047] newInstance(Properties properties, ClassIoader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in JCA80047]	:A80043]
Obtains a object implementing the SCAClientFactory class, using a sReturns: • object which implements the SCAClientFactory class The SCAClientFactory.newInstance(Properties. URI) method Mimplements the SCAClientFactory class for the SCA Domain idea [JCA80044] Parameters: • properties - a set of Properties that can be used when creating the SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Properties, URI) method MinoSuchDomainException if the domainURI parameter does not in JCA80045] mewInstance(ClassIoader classLoader, URI domainURI) method MinoSuchDomainException if the SCAClientFactory class using a signerur	
Returns: • object which implements the SCAClientFactory class The SCAClientFactory.newInstance(Properties, URI) method Mimplements the SCAClientFactory class for the SCA Domain ider [JCA80044] Parameters: • properties - a set of Properties that can be used when creating to SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Properties, URI) method Minstance(Properties, URI) method Minstance(Plassocial Plass	
 object which implements the SCAClientFactory class The SCAClientFactory.newInstance(Properties, URI) method Mimplements the SCAClientFactory class for the SCA Domain ident [JCA80044] Parameters: properties - a set of Properties that can be used when creating the SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: 	specified set of properties.
The SCAClientFactory.newInstance(Properties, URI) method M implements the SCAClientFactory class for the SCA Domain idea [JCA80044] Parameters: • properties - a set of Properties that can be used when creating the SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Properties, URI) method M NoSuchDomainException if the domainURI parameter does not in [JCA80045] mewInstance(ClassIoader classLoader, URI domainURI) method M NoSuchDomainException if the SCAClientFactory class using a signerur signer which implements the SCAClientFactory class using a signerur signer	
implements the SCAClientFactory class for the SCA Domain ider [JCA80044] Parameters: • properties - a set of Properties that can be used when creating the SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Properties, URI) method M NoSuchDomainException if the domainURI parameter does not in JCA80045] newInstance(ClassIoader classLoader, URI domainURI) method. Obtains a object implementing the SCAClientFactory class using a signature. • object which implements the SCAClientFactory class. The SCAClientFactory.newInstance(ClassIoader, URI) method implements the SCAClientFactory class for the SCA Domain ider JJCA80046] Parameters: • classLoader - a ClassLoader to use when creating the object with SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(ClassIoader, URI) method NoSuchDomainException if the domainURI parameter does not in JCA80047] newInstance(Properties properties, ClassIoader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in JCA80047]	
Parameters: properties - a set of Properties that can be used when creating to SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(Properties, URI) method M NoSuchDomainException if the domainURI parameter does not in JCA80045] mewInstance(ClassIoader classLoader, URI domainURI) method. Obtains a object implementing the SCAClientFactory class using a serious with simplements the SCAClientFactory class. object which implements the SCAClientFactory class. The SCAClientFactory.newInstance(ClassIoader, URI) method implements the SCAClientFactory class for the SCA Domain idea (JCA80046) Parameters: classLoader - a ClassLoader to use when creating the object with SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(ClassIoader, URI) method NoSuchDomainException if the domainURI parameter does not in JCA80047] mewInstance(Properties properties, ClassIoader classLoader, URI) mewInstance(Properties properties, ClassIoader, URI) mewInstance(Properties properties, ClassIoader, URI) mewInstance(Properties properties pro	
 properties - a set of Properties that can be used when creating to SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(Properties, URI) method M NoSuchDomainException if the domainURI parameter does not in [JCA80045] newInstance(ClassIoader classLoader, URI domainURI) method [JCA80045] newInstance(ClassIoader classLoader, URI domainURI) method [JCA80045] object which implements the SCAClientFactory class using a signature. object which implements the SCAClientFactory class [JCA80046] Parameters: classLoader - a ClassLoader to use when creating the object who SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(ClassIoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, ClassIoader classLoader, UI Obtains a object implementing the SCAClientFactory class using a specified classIoader. Returns: 	nuneu by the domainors para
 properties - a set of Properties that can be used when creating to SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(Properties, URI) method M NoSuchDomainException if the domainURI parameter does not in [JCA80045] newInstance(ClassIoader classLoader, URI domainURI) method [JCA80045] newInstance(ClassIoader classLoader, URI domainURI) method [JCA80045] object which implements the SCAClientFactory class using a signature. object which implements the SCAClientFactory class [JCA80046] Parameters: classLoader - a ClassLoader to use when creating the object who SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(ClassIoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, ClassIoader classLoader, UI Obtains a object implementing the SCAClientFactory class using a specified classIoader. Returns: 	
SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(Properties, URI) method M NoSuchDomainException if the domainURI parameter does not in [JCA80045] newInstance(ClassIoader classLoader, URI domainURI) method Obtains a object implementing the SCAClientFactory class using a signeturns: object which implements the SCAClientFactory class using a signeturns: object which implements the SCAClientFactory class The SCAClientFactory.newInstance(ClassIoader, URI) method implements the SCAClientFactory class for the SCA Domain idea [JCA80046] Parameters: classLoader - a ClassLoader to use when creating the object with SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(ClassIoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, ClassIoader classLoader, UI Obtains a object implementing the SCAClientFactory class using a sispecified classIoader. Returns:	the object which implements the
Exceptions: The SCAClientFactory.newInstance(Properties, URI) method M NoSuchDomainException if the domainURI parameter does not in [JCA80045] mewInstance(Classloader classLoader, URI domainURI) method [JCA80045] mewInstance(Classloader classLoader, URI domainURI) method [JCA80045] newInstance(Classloader classLoader, URI domainURI) method [JCA80045] Parameters: ClassLoader - a ClassLoader to Use when creating the object who SCAClientFactory class [JCA80046] Parameters: ClassLoader - a ClassLoader to Use when creating the object who SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] mewInstance(Properties properties, Classloader classLoader, URI) method NosuchDomainException if the SCAClientFactory class using a signerified classloader. Returns:	
 The SCAClientFactory.newInstance(Properties, URI) method M NoSuchDomainException if the domainURI parameter does not in [JCA80045] newInstance(Classloader classLoader, URI domainURI) method. Obtains a object implementing the SCAClientFactory class using a signeturns: object which implements the SCAClientFactory class. The SCAClientFactory.newInstance(Classloader, URI) method implements the SCAClientFactory class for the SCA Domain idea [JCA80046] Parameters: classLoader - a ClassLoader to use when creating the object whe SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method parameter does not in [JCA80047] Obtains a object implementing the SCAClientFactory class using a signecified classloader. Returns:	e returned SCAClient object
NoSuchDomainException if the domainURI parameter does not in [JCA80045] newInstance(ClassIoader classLoader, URI domainURI) method. Obtains a object implementing the SCAClientFactory class using a signeturns: • object which implements the SCAClientFactory class The SCAClientFactory.newInstance(ClassIoader, URI) method implements the SCAClientFactory class for the SCA Domain idea [JCA80046] Parameters: • classLoader - a ClassLoader to use when creating the object whe SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(ClassIoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, ClassIoader classLoader, URI) method NosuchDomainException if the SCAClientFactory class using a signed classIoader. Returns:	
newInstance(ClassIoader classLoader, URI domainURI) method. Obtains a object implementing the SCAClientFactory class using a singletime. • object which implements the SCAClientFactory class The SCAClientFactory.newInstance(ClassIoader, URI) method implements the SCAClientFactory class for the SCA Domain ider [JCA80046] Parameters: • classLoader - a ClassLoader to use when creating the object with SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(ClassIoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] mewInstance(Properties properties, ClassIoader classLoader, URI) method NosuchDomainException if the SCAClientFactory class using a signed classIoader. Returns:	
newInstance(ClassIoader classLoader, URI domainURI) method. Obtains a object implementing the SCAClientFactory class using a since the scaclient of the scalled o	identify a valid SCA Domain.
Obtains a object implementing the SCAClientFactory class using a speturns: • object which implements the SCAClientFactory class The SCAClientFactory.newInstance(Classloader, URI) method implements the SCAClientFactory class for the SCA Domain idel [JCA80046] Parameters: • classLoader - a ClassLoader to use when creating the object whe SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] mewInstance(Properties properties, Classloader classLoader, URI) MewInstance(Properties properties pr	
Obtains a object implementing the SCAClientFactory class using a speturns: • object which implements the SCAClientFactory class The SCAClientFactory.newInstance(Classloader, URI) method implements the SCAClientFactory class for the SCA Domain idel [JCA80046] Parameters: • classLoader - a ClassLoader to use when creating the object whe SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] mewInstance(Properties properties, Classloader classLoader, URI) MewInstance(Properties properties pr	
Returns: • object which implements the SCAClientFactory class The SCAClientFactory.newInstance(Classloader, URI) method implements the SCAClientFactory class for the SCA Domain ider [JCA80046] Parameters: • classLoader - a ClassLoader to use when creating the object whe SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] mewInstance(Properties properties, Classloader classLoader, URI) mewInstance(Properties properties pr	
 object which implements the SCAClientFactory class The SCAClientFactory.newInstance(Classloader, URI) method implements the SCAClientFactory class for the SCA Domain idel [JCA80046] Parameters: classLoader - a ClassLoader to use when creating the object whe SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] mewInstance(Properties properties, Classloader classLoader, URI) Dotains a object implementing the SCAClientFactory class using a supposition of classloader. Returns: 	poomed diacologaett.
The SCAClientFactory.newInstance(Classloader, URI) method implements the SCAClientFactory class for the SCA Domain idel [JCA80046] Parameters: • classLoader - a ClassLoader to use when creating the object whe SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] mewInstance(Properties properties, Classloader classLoader, URI) method NosuchDomainException if the domainURI parameter does not in [JCA80047] Dotains a object implementing the SCAClientFactory class using a suspecified classloader. Returns:	
implements the SCAClientFactory class for the SCA Domain ider [JCA80046] Parameters: • classLoader - a ClassLoader to use when creating the object whe SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] mewInstance(Properties properties, Classloader classLoader, URI) Dotains a object implementing the SCAClientFactory class using a supposition of classloader. Returns:	MUST return an object which
Parameters: • classLoader - a ClassLoader to use when creating the object whe SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] mewInstance(Properties properties, Classloader classLoader, URI) MewInstance(Properties properties pr	
 classLoader - a ClassLoader to use when creating the object whe SCAClientFactory class. domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] 	
SCAClientFactory class. • domainURI - a URI for the SCA Domain which is targeted by the Exceptions: • The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047]	
 domainURI - a URI for the SCA Domain which is targeted by the Exceptions: The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] 	nich implements the
Exceptions: The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not i [JCA80047] newInstance(Properties properties, Classloader classLoader, Uii Obtains a object implementing the SCAClientFactory class using a specified classloader. Returns:	returned SCAClient chiest
The SCAClientFactory.newInstance(Classloader, URI) method NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classloader classLoader, URI) Obtains a object implementing the SCAClientFactory class using a supposition of classloader. Returns:	Freturied SCACIIENT Object
NoSuchDomainException if the domainURI parameter does not in [JCA80047] newInstance(Properties properties, Classical classica	MLICT throw o
newInstance(Properties properties, Classloader classLoader, Ul Obtains a object implementing the SCAClientFactory class using a specified classloader. Returns:	
Obtains a object implementing the SCAClientFactory class using a specified classloader. Returns:	
Obtains a object implementing the SCAClientFactory class using a specified classloader. Returns:	
specified classloader. Returns:	RI domainURI) method:
Returns:	pecified set of properties and a
A PROOF Which implements the SUM Heathertery close	
The SCAClientFactory.newInstance(Properties, Classloader, UF which implements the SCAClientFactory class for the SCA Doma	N.) as a thread MILIOT

1964 Parameters: 1965 properties - a set of Properties that can be used when creating the object which implements the SCAClientFactory class. 1966 classLoader - a ClassLoader to use when creating the object which implements the 1967 1968 SCAClientFactory class. 1969 domainURI - a URI for the SCA Domain which is targeted by the returned SCAClient object 1970 Exceptions: 1971 The SCAClientFactory.newInstance(Properties, Classloader, URI) MUST throw a 1972 NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain. [JCA80049] 1973 1974 1975 getService(Class<T> interfaze, String serviceURI) method: 1976 Obtains a proxy reference object for a specified target service in a specified SCA Domain. 1977 Returns: 1978 <T> a proxy object which implements the business interface T 1979 Invocations of a business method of the proxy causes the invocation of the corresponding operation 1980 of the target service. 1981 The SCAClientFactory.getService method MUST return a proxy object which implements the 1982 business interface defined by the interfaze parameter and which can be used to invoke operations on 1983 the service identified by the serviceURI parameter. [JCA80050] 1984 Parameters: 1985 interfaze - a Java interface class which is the business interface of the target service 1986 serviceURI - a String containing the relative URI of the target service within its SCA Domain. 1987 Takes the form componentName/serviceName or can also take the extended form componentName/serviceName/bindingName to use a specific binding of the target service 1988 1989 **Exceptions:** The SCAClientFactory.getService method MUST throw a NoSuchServiceException if a service with 1990 he relative URI serviceURI and a business interface which matches interfaze cannot be found in the 1991 SCA Domain targeted by the SCAClient object. [JCA80051] 1992 1993 The SCAClientFactory.getService method MUST throw a NoSuchServiceException if the domainURI of the SCAClientFactory does not identify a valid SCA Domain. [JCA80052] 1994 1995 1996 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 1997 1998 **SCAClientFactory** 1999 2000 getDomainURI() method: 2001 Obtains the Domain URI value for this SCAClientFactory 2002 2003 URI of the target SCA Domain for this SCAClientFactory The SCAClientFactory.getDomainURI method MUST return the SCA Domain URI of the Domain 2004 associated with the SCAClientFactory object. [JCA80053] 2005 2006 Parameters: 2007 none

2008

Exceptions:

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

2010 2011 2012

2013

2009

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.

2014 2015 2016

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.

2018 2019 2020

> 2021 2022

2023

2017

9.11 SCAClientFactoryFinder Interface

public interface SCAClientFactoryFinder {

SCAClientFactory find(Properties properties,

throws NoSuchDomainException ;

package org.oasisopen.sca.client;

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:

ClassLoader classLoader,

URI domainURI)

2024 2025

```
2026
2027
2028
2029
```

2030 2031

2032 2033 2034

2035

2037

2036

2040

2041

2042 2043

2047

2048

2049

2052

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

2038 Obtains an implementation of the SCAClientFactory interface. 2039

SCAClientFactory implementation object

Figure 9-10: SCAClientFactoryFinder Interface

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

2044 Parameters:

2045 2046

- 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

2050 Exceptions:

2051

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

9.12 SCAClientFactoryFinderImpl Class 2053

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.

2074

2054 2055

2056 2057 2058

2059 2060

2061 2062

2063 2064

2065

2066

2067

2068 2069 2070

2071 2072

2073

2075

2076

2079

2080 2081

2082

2083

2084

2085

2086

2087

2089

2090

2091

2092 2093

2094

2097

SCAClientFactoryFinderImpI which implements the SCAClientFactoryFinder interface

Parameters:

2077 none

2078 **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:

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

2088 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.
- 2095 domainURI - a URI for the SCA Domain targeted by the SCAClientFactory

2096 Exceptions:

ServiceRuntimeException - if the SCAClientFactory implementation could not be found

9.13 NoSuchDomainException

20982099

2100 2101

2102 2103

2104 2105 2106

2107

2108

2109

211021112112

2113 2114

2115 2116

2117

21182119

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.

9.14 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.

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.

improperly used, the SCA runtime MUST NOT run the component which uses the invalid implementation

2127 code. [JCA90001]

2120 2121

2122

2123 2124

2125

2126

2128

2129 2130

2131

213221332134

2135 2136

2137

2138

2139 2140

2141

2142

2143 2144

2145

2146

2147 2148

2149

SCA annotations MUST NOT be used on static methods or on static fields. It is an error to use an SCA annotation on a static method or a static field of an implementation class and the SCA runtime MUST

NOT instantiate such an implementation class. [JCA90002]

10.1 @AllowsPassByReference

<u>Figure</u> 10-1 defines the **@**AllowsPassByReference annotation:

Deleted: The following Java code

Field Code Changed

Field Code Changed

```
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, METHOD, FIELD, PARAMETER})
@Retention(RUNTIME)
public @interface AllowsPassByReference {
   boolean value() default true;
}
```

Figure 10-1: AllowsPassByReference Annotation

2154

2155

2156

2157

2158

2159

2160

2161

2162

2163 2164 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

Field Code Changed

Formatted: Bullets and Numbering

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]

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010

Page 60 of 126

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

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

Deleted: ¶
The following snippet

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

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

Deleted: ¶
The following snippet

<u>Snippet</u> 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 java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Inherited
@Target({METHOD})
@Retention(RUNTIME)
public @interface AsyncFault {
        Class<?>[] value() default {};
}
```

sca-javacaa-1.1-spec-<u>cd04</u> Copyright © OASIS® 2005, 2010. All Rights Reserved. 06 Feb 2010 Page 61 of 126 The @AsyncFault annotation is used to indicate the faults/exceptions which are returned by the asynchronous service method which it annotates.

10.3 @AsyncInvocation

<u>Figure 10-3 defines the @AsyncInvocation</u> annotation, which is used to attach the "asyncInvocation" policy intent to an interface or to a method:

```
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
```

```
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:

```
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
```

```
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 {
```

```
2268
               String AUTHENTICATION = SCA_PREFIX + "authentication";
2269
               String AUTHENTICATION_MESSAGE = AUTHENTICATION + ".message";
2270
               String AUTHENTICATION_TRANSPORT = AUTHENTICATION + ".transport";
2271
2272
2273
                * List of authentication qualifiers (such as "message"
2274
                * or "transport").
2275
2276
                 * @return authentication qualifiers
                */
2277
2278
               @Qualifier
2279
               String[] value() default "";
2280
```

Figure 10-4: Authentication Annotation

2281

2282 2283

2284

2285

22862287

2288 2289

2290 2291

2292 2293 2294

2295

2296

2297 2298

2299

2300

2301 2302

2303 2304

2305 2306

2307

2308

2309

2310

2311

23122313

2314 2315

2316 2317

2318

2319

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:

```
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

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

<u>Figure</u> 10-6 defines the @*Callback* annotation:

Deleted: The following Java code

Deleted: invocation requires

Deleted:

Deleted: and

sca-javacaa-1.1-spec-<u>cd04</u> Copyright © OASIS® 2005, <u>2010.</u> All Rights Reserved. 06 Feb 2010 Page 63 of 126

```
2320
2321
             package org.oasisopen.sca.annotation;
2322
2323
             import static java.lang.annotation.ElementType.FIELD;
2324
             import static java.lang.annotation.ElementType.METHOD;
             import static java.lang.annotation.ElementType.TYPE;
2325
2326
             import static java.lang.annotation.RetentionPolicy.RUNTIME;
2327
             import java.lang.annotation.Retention;
2328
             import java.lang.annotation.Target;
2329
2330
             @Target({TYPE, METHOD, FIELD})
2331
             @Retention(RUNTIME)
2332
             public @interface Callback {
2333
2334
                Class<?> value() default Void.class;
2335
2336
          Figure 10-6: Callback Annotation
2337
2338
        The @Callback annotation is used to annotate a service interface or to annotate a Java class (used to
2339
        define an interface) with a callback interface by specifying the Java class object of the callback interface
2340
        as an attribute.
                                                                                                                      Deleted: following
2341
        The @Callback annotation has the attribute:
2342
            value - the name of a Java class file containing the callback interface
2343
        The @Callback annotation can also be used to annotate a method or a field of an SCA implementation.
                                                                                                                      Deleted: ¶
2344
        class, in order to have a callback object injected. When used to annotate a method or a field of an
2345
        implementation class for injection of a callback object, the @Callback annotation MUST NOT specify an
         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
2346
2347
        bidirectional service offered by the implementation class. [JCA90054] When used to annotate a setter
2348
2349
        method or a field of an implementation class for injection of a callback object, the SCA runtime MUST
2350
        inject a callback reference proxy into that method or field when the Java class is initialized, if the
2351
        component is invoked via a service which has a callback interface and where the type of the setter
2352
        method or field corresponds to the type of the callback interface. [JCA90058]
2353
        The @Callback annotation MUST NOT appear on a setter method or a field of a Java implementation
                                                                                                                      Deleted: An
2354
        class that has COMPOSITE scope. [JCA90057]
                                                                                                                      Deleted: follows:
2355
        Snippet 10-3 shows an example use of the @Callback annotation to declare a callback interface.
2356
2357
             package somepackage;
2358
             import org.oasisopen.sca.annotation.Callback;
2359
             import org.oasisopen.sca.annotation.Remotable;
2360
             @Remotable
2361
             @Callback(MyServiceCallback.class)
2362
             public interface MyService {
2363
2364
                  void someMethod(String arg);
2365
2366
2367
             @Remotable
2368
             public interface MyServiceCallback {
2369
2370
                  void receiveResult(String result);
2371
                                                                                                                      Deleted: ¶
2372
                                                                                                                      In this example, the
          Snippet 10-3: Use of @Callback
```

06 Feb 2010

Page 64 of 126

2373

sca-iavacaa-1.1-spec-cd04

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

```
Deleted: :
2374
       The implied component type is for Snippet 10-3 is shown in Snippet 10-4.
2375
2376
            <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
                                                                                                             Deleted: 200903
2377
2378
               <service name="MvService">
2379
                 _<interface.java interface="somepackage.MyService"
2380
                                callbackInterface="somepackage.MyServiceCallback"/>
2381
               </service>
2382
            </componentType>
2383
         Snippet 10-4: Implied componentType for Snippet 10-3
         10.7 @ComponentName
2384
                                                                                                             Deleted: The following Java code
2385
       Figure 10-7 defines the @ComponentName annotation:
2386
2387
            package org.oasisopen.sca.annotation;
2388
2389
            import static java.lang.annotation.ElementType.FIELD;
2390
            import static java.lang.annotation.ElementType.METHOD;
2391
            import static java.lang.annotation.RetentionPolicy.RUNTIME;
                                                                                                             Deleted: ElementType.TYPE
2392
            import java.lang.annotation.Retention;
2393
            import java.lang.annotation.Target;
2394
2395
            @Target({METHOD, FIELD})
2396
            @Retention(RUNTIME)
2397
            public @interface ComponentName {
2398
2399
2400
         Figure 10-7: ComponentName Annotation
2401
2402
       The @ComponentName annotation is used to denote a Java class field or setter method that is used to
2403
       inject the component name.
                                                                                                             Deleted: The following snippet
       <u>Snippet</u> 10-5 shows a component name field definition sample.
2404
2405
2406
            @ComponentName
2407
            private String componentName;
                                                                                                             Deleted: ¶
2408
                                                                                                             The following snippet
         Snippet 10-5: Use of @ComponentName on a Field
2409
       Snippet 10-6 shows a component name setter method sample.
2410
2411
2412
            @ComponentName
2413
            public void setComponentName(String name) {
2414
2415
         Snippet 10-6: Use of @ComponentName on a Setter
2416
         10.8 @Confidentiality
2417
                                                                                                             Deleted: The following Java code
2418
       <u>Figure</u> 10-8 defines the @Confidentiality annotation:
2419
```

06 Feb 2010

Page 65 of 126

sca-javacaa-1.1-spec-cd04

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

```
2420
           package org.oasisopen.sca.annotation;
2421
2422
           import static java.lang.annotation.ElementType.FIELD;
2423
           import static java.lang.annotation.ElementType.METHOD;
2424
           import static java.lang.annotation.ElementType.PARAMETER;
2425
           import static java.lang.annotation.ElementType.TYPE;
2426
           import static java.lang.annotation.RetentionPolicy.RUNTIME;
2427
           import static org.oasisopen.sca.Constants.SCA_PREFIX;
2428
2429
           import java.lang.annotation.Inherited;
2430
           import java.lang.annotation.Retention;
2431
           import java.lang.annotation.Target;
2432
2433
           @Inherited
2434
           @Target({TYPE, FIELD, METHOD, PARAMETER})
2435
           @Retention(RUNTIME)
2436
           @Intent(Confidentiality.CONFIDENTIALITY)
2437
           public @interface Confidentiality {
2438
               String CONFIDENTIALITY = SCA_PREFIX + "confidentiality";
2439
               String CONFIDENTIALITY_MESSAGE = CONFIDENTIALITY + ".message";
2440
               String CONFIDENTIALITY_TRANSPORT = CONFIDENTIALITY + ".transport";
2441
2442
2443
                * List of confidentiality qualifiers such as "message" or
2444
                 * "transport".
2445
2446
                 * @return confidentiality qualifiers
2447
2448
               @Qualifier
2449
               String[] value() default "";
2450
```

Figure 10-8: Confidentiality Annotation

2455

2456

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.

Deleted: that

Deleted: invocation requires

Deleted: ¶

Deleted: and details

Deleted: The following Java code

10.9 @Constructor

<u>Figure</u> 10-9 defines the @Constructor annotation:

2457 2458 2459

2460 2461

2462

2463

2464

2465 2466

2467

2468

```
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

2469 2470 2471

2472

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

Field Code Changed

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 66 of 126

```
and the constructor has parameters, each of these parameters MUST have either a @Property
2473
2474
        annotation or a @Reference annotation. [JCA90003]
                                                                                                               Deleted: The following snippet
2475
        Snippet 10-7 shows a sample for the @Constructor annotation.
2476
2477
            public class HelloServiceImpl implements HelloService {
2478
2479
               public HelloServiceImpl(){
2480
2481
2482
2483
                @Constructor
2484
                public HelloServiceImpl(@Property(name="someProperty")
2485
                                          String someProperty ){
2486
2487
2488
2489
                public String hello(String message) {
2490
2491
2492
2493
          Snippet 10-7: Use of @Constructor
          10.10 @Context
2494
                                                                                                               Deleted: The following Java code
2495
        Figure 10-10 defines the @Context annotation:
2496
2497
            package org.oasisopen.sca.annotation;
2498
2499
            import static java.lang.annotation.ElementType.FIELD;
2500
            import static java.lang.annotation.ElementType.METHOD;
2501
            import static java.lang.annotation.RetentionPolicy.RUNTIME;
2502
            import java.lang.annotation.Retention;
2503
            import java.lang.annotation.Target;
2504
2505
            @Target({METHOD, FIELD})
2506
            @Retention(RUNTIME)
2507
            public @interface Context {
2508
2509
2510
          Figure 10-10: Context Annotation
2511
        The @Context annotation is used to denote a Java class field or a setter method that is used to inject a
2512
2513
        composite context for the component. The type of context to be injected is defined by the type of the Java
2514
        class field or type of the setter method input argument; the type is either ComponentContext or
2515
        RequestContext.
2516
        The @Context annotation has no attributes.
                                                                                                               Deleted: The following snippet
2517
        Snippet 10-8 shows a ComponentContext field definition sample.
2518
2519
            @Context
2520
            protected ComponentContext context;
                                                                                                               Deleted: ¶
2521
                                                                                                               The following snippet
          Snippet 10-8: Use of @Context for a ComponentContext
```

06 Feb 2010

Page 67 of 126

2522

sca-iavacaa-1.1-spec-cd04

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

```
2523
        Snippet 10-9 shows a RequestContext field definition sample.
2524
2525
            @Context
2526
            protected RequestContext context;
2527
          Snippet 10-9: Use of @Context for a RequestContext
          10.11 @Destroy
2528
2529
2530
            The @Destroy annotation is used to denote a single Java class method that will be called when the
2531
            <u>scope defined for the implementation class ends. <mark>A method annotated with @Destroy can have</mark></u>
2532
            any access modifier and MUST have a void return type and no arguments. [JCA90004]
2533
            If there is a method annotated with @Destroy that matches the criteria for the annotation, the
2534
            SCA runtime MUST call the annotated method when the scope defined for the implementation
            class ends. [JCA90005]
2535
2536
            The following snippet shows a sample for a destroy method definition.
2537
        Figure 10-11 defines the @Destroy annotation:
2538
2539
            package org.oasisopen.sca.annotation;
2540
2541
            import static java.lang.annotation.ElementType.METHOD;
2542
             import static java.lang.annotation.RetentionPolicy.RUNTIME;
2543
            import java.lang.annotation.Retention;
2544
            import java.lang.annotation.Target;
2545
2546
            @Target(METHOD)
2547
            @Retention(RUNTIME)
2548
            public @interface Destroy {
2549
2550
2551
2552
            The @Destroy annotation is used to denote a single Java class method that will be called when the
2553
            scope defined for the implementation class ends. A method annotated with @Destroy can have
2554
            any access modifier and MUST have a void return type and no arguments. [JCA90004]
2555
            If there is a method annotated with @Destroy that matches the criteria for the annotation, the
2556
            SCA runtime MUST call the annotated method when the scope defined for the implementation
2557
            class ends. [JCA90005]
2558
            The following snippet shows a sample for a destroy method definition.
2559
          Figure 10-11: Destroy Annotation
2560
2561
        The @Destroy annotation is used to denote a single Java class method that will be called when the scope
2562
        defined for the implementation class ends. A method annotated with @Destroy can have any access
2563
        modifier and MUST have a void return type and no arguments. [JCA90004]
2564
        If there is a method annotated with @Destroy that matches the criteria for the annotation, the SCA
2565
        runtime MUST call the annotated method when the scope defined for the implementation class ends.
        [JCA90005]
2566
2567
        Snippet 10-10 shows a sample for a destroy method definition.
2568
2569
            @Destrov
```

public void myDestroyMethod() {

2570

Deleted: The following Java code

```
2571
2572
2573
         Snippet 10-10: Use of @Destroy
         10.12 @EagerInit
2574
                                                                                                              Deleted: The following Java code
       Figure 10-12: EagerInit Annotation defines the @EagerInit annotation:
2575
2576
2577
            package org.oasisopen.sca.annotation;
2578
2579
            import static java.lang.annotation.ElementType.TYPE;
2580
            import static java.lang.annotation.RetentionPolicy.RUNTIME;
2581
            import java.lang.annotation.Retention;
2582
            import java.lang.annotation.Target;
2583
2584
            @Target(TYPE)
2585
            @Retention(RUNTIME)
2586
            public @interface EagerInit {
2587
2588
2589
         Figure 10-12: EagerInit Annotation
2590
2591
        The @EagerInit annotation is used to mark the Java class of a COMPOSITE scoped implementation for
2592
       eager initialization. When marked for eager initialization with an @EagerInit annotation, the composite
                                                                                                              Field Code Changed
            ed instance MUST be created when its containing component is started. [JCA90007]
2593
         10.13 @Init
2594
2595
                                                                                                              Deleted: The following Java code
2596
           The @Init annotation is used to denote a single Java class method that is called when the scope
2597
           defined for the implementation class starts. A method marked with the @Init annotation can have
2598
           any access modifier and MUST have a void return type and no arguments. [JCA90008]
2599
           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.
2600
2601
2602
           The following snippet shows an example of an init method definition.
2603
       Figure 10-13: Init Annotation defines the @Init annotation:
2604
2605
            package org.oasisopen.sca.annotation;
2606
2607
            import static java.lang.annotation.ElementType.METHOD;
2608
            import static java.lang.annotation.RetentionPolicy.RUNTIME;
2609
            import java.lang.annotation.Retention;
2610
            import java.lang.annotation.Target;
2611
2612
            @Target(METHOD)
2613
            @Retention(RUNTIME)
2614
            public @interface Init {
2615
2616
2617
```

2618

```
2619
            The @Init annotation is used to denote a single Java class method that is called when the scope
2620
            defined for the implementation class starts. A method marked with the @Init annotation can have
2621
             any access modifier and MUST have a void return type and no arguments. [JCA90008]
2622
            If there is a method annotated with @Init that matches the criteria for the annotation, the SCA
2623
             <u>runtime MUST call the annotated method after all property and reference injection is complete.</u>
2624
            [JCA90009]
2625
            The following snippet shows an example of an init method definition.
2626
          Figure 10-13: Init Annotation
2627
2628
        The @Init annotation is used to denote a single Java class method that is called when the scope defined
2629
        for the implementation class starts. A method marked with the @Init annotation can have any access
2630
        modifier and MUST have a void return type and no arguments. [JCA90008]
2631
        If there is a method annotated with @Init that matches the criteria for the annotation, the SCA runtime
2632
         MUST call the annotated method after all property and reference injection is complete. [JCA90009]
```

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

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

Snippet 10-11: Use of @Init

2633

2634 2635

2636

2637 2638 2639

2640

2641 2642

10.14 @Integrity

<u>Figure</u> 10-14 defines the @*Integrity* annotation:

Deleted: The following Java code

```
2643
           package org.oasisopen.sca.annotation;
2644
2645
           import static java.lang.annotation.ElementType.FIELD;
2646
           import static java.lang.annotation.ElementType.METHOD;
2647
           import static java.lang.annotation.ElementType.PARAMETER;
2648
           import static java.lang.annotation.ElementType.TYPE;
2649
           import static java.lang.annotation.RetentionPolicy.RUNTIME;
2650
           import static org.oasisopen.sca.Constants.SCA_PREFIX;
2651
2652
           import java.lang.annotation.Inherited;
2653
           import java.lang.annotation.Retention;
2654
           import java.lang.annotation.Target;
2655
2656
           @Inherited
2657
           @Target({TYPE, FIELD, METHOD, PARAMETER})
2658
           @Retention(RUNTIME)
2659
           @Intent(Integrity.INTEGRITY)
2660
           public @interface Integrity {
2661
               String INTEGRITY = SCA_PREFIX + "integrity";
2662
               String INTEGRITY_MESSAGE = INTEGRITY + ".message";
2663
               String INTEGRITY_TRANSPORT = INTEGRITY + ".transport";
2664
2665
2666
                * List of integrity qualifiers (such as "message" or "transport").
2667
2668
                * @return integrity qualifiers
2669
2670
               @Oualifier
2671
               String[] value() default "";
```

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 70 of 126 2672

Figure 10-14: Integrity Annotation

2673 2674

2678

2679

2680

2681 2682

2683 2684

2685

2686

2687

2688 2689

2690

2691

2692 2693

2694

2695

2696

2697

2698 2699 2700

2701

2702 2703

2704 2705 2706

2707

2708 2709

2710 2711

2675 2676 2677

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.

Deleted: ¶

Deleted: and details

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 "";
     \mbox{\scriptsize \star} The XML namespace for the intent.
     * @return the XML namespace for the intent
    String targetNamespace() default "";
     * The name of the intent within its namespace.
     * @return name of the intent within its namespace
    String localPart() default "";
```

Figure 10-15: Intent Annotation

2712 2713 2714

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.

2715 See the section "How to Create Specific Intent Annotations" for details and samples of how to define new 2716 intent annotations.

10.16 @ManagedSharedTransaction

Figure 10-16 defines the @ManagedSharedTransaction annotation:

2719 2720 2721

2717

2718

package org.oasisopen.sca.annotation;

sca-iavacaa-1.1-spec-cd04 Copyright © OASIS® 2005, 2010. All Rights Reserved. 06 Feb 2010

Page 71 of 126

```
2722
           import static java.lang.annotation.ElementType.FIELD;
2723
           import static java.lang.annotation.ElementType.METHOD;
2724
            import static java.lang.annotation.ElementType.PARAMETER;
2725
            import static java.lang.annotation.ElementType.TYPE;
2726
           import static java.lang.annotation.RetentionPolicy.RUNTIME;
2727
            import static org.oasisopen.sca.Constants.SCA_PREFIX;
2728
2729
            import java.lang.annotation.Inherited;
2730
           import java.lang.annotation.Retention;
2731
           import java.lang.annotation.Target;
2732
2733
2734
            * The @ManagedSharedTransaction annotation is used to indicate that
2735
            * a distributed ACID transaction is required.
2736
2737
           @Inherited
2738
           @Target({TYPE, FIELD, METHOD, PARAMETER})
2739
           @Retention(RUNTIME)
2740
            @Intent(ManagedSharedTransaction.MANAGEDSHAREDTRANSACTION)
2741
           public @interface ManagedSharedTransaction {
2742
               String MANAGEDSHAREDTRANSACTION = SCA_PREFIX + "managedSharedTransaction";
2743
```

Figure 10-16: ManagedSharedTransaction Annotation

2744

2745 2746

2747

2748

2749

2750 2751

2752 2753

2754

2755

2756

2757

2758

2759 2760

2761

2762

2763 2764 2765

2766

2767 2768 2769

2770

2771

2772

2773

2774 2775

2776 2777 2778 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;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import static org.oasisopen.sca.Constants.SCA_PREFIX;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
 * The @ManagedTransaction annotation is used to indicate the
 * need for an ACID transaction environment.
@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 "";
}
```

Figure 10-17: ManagedTransaction Annotation

The @ManagedTransaction annotation is used to indicate the need for 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.

10.18 @MutualAuthentication

Figure 10-18 defines the @MutualAuthentication 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 @MutualAuthentication annotation is used to indicate that
^* a mutual authentication policy is needed. ^*/
@Inherited
@Target({TYPE, FIELD, METHOD, PARAMETER})
@Retention(RUNTIME)
@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;
```

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 73 of 126

```
2830
           import static java.lang.annotation.ElementType.PARAMETER;
2831
           import static java.lang.annotation.ElementType.TYPE;
2832
            import static java.lang.annotation.RetentionPolicy.RUNTIME;
2833
            import static org.oasisopen.sca.Constants.SCA_PREFIX;
2834
2835
            import java.lang.annotation.Inherited;
2836
            import java.lang.annotation.Retention;
2837
           import java.lang.annotation.Target;
2838
2839
2840
            * The @NoManagedTransaction annotation is used to indicate that
2841
            * a non-transactional environment is needed.
2842
2843
           @Inherited
2844
           @Target({TYPE, FIELD, METHOD, PARAMETER})
2845
            @Retention(RUNTIME)
2846
           @Intent(NoManagedTransaction.NOMANAGEDTRANSACTION)
2847
           public @interface NoManagedTransaction \{
2848
               String NOMANAGEDTRANSACTION = SCA_PREFIX + "noManagedTransaction";
2849
```

Figure 10-19: NoManagedTransaction Annotation

2851 2852 2853

2854

2855

2856

2857 2858 2859

2860 2861

2862

2863

2864

2865 2866

2867

2868

2869 2870 2871

2850

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:

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 OneWay {
```

Figure 10-20: OneWay Annotation

2872 2873 2874

2875

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

2876 2877 2878 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]

2879

The @OneWay annotation has no attributes.

2880 2881 Snippet 10-12 shows the use of the @OneWay annotation on an interface.

Deleted: The following snippet

The @OneWay annotation is

method to indicate that invocations will be dispatched in

a non-blocking fashion as

described in the section on Asynchronous Programming.¶

used on a Java interface or class

Deleted: ¶

sca-javacaa-1.1-spec-cd04 Copyright © OASIS® 2005, 2010. All Rights Reserved. 06 Feb 2010 Page 74 of 126

```
2882 package services.hello;
2883
2884 import org.oasisopen.sca.annotation.OneWay;
2885
2886 public interface HelloService {
2887 @OneWay
2888 void hello(String name);
}
2890 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 "";
}
```

Figure 10-21: PolicySets Annotation

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:

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

@Target({METHOD, FIELD, PARAMETER})
@Retention(RUNTIME)
```

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 75 of 126

```
2934
             public @interface Property {
2935
2936
                String name() default "";
2937
                boolean required() default true;
2938
2939
          Figure 10-22: Property Annotation
2940
2941
        The @Property annotation is used to denote a Java class field, a setter method, or a constructor
2942
        parameter that is used to inject an SCA property value. The type of the property injected, which can be a
2943
        simple Java type or a complex Java type, is defined by the type of the Java class field or the type of the
2944
        input parameter of the setter method or constructor.
2945
        When the Java type of a field, setter method or constructor parameter with the @Property annotation is a primitive
                                                                                                                       Field Code Changed
        type or a JAXB annotated class, the SCA runtime MUST convert a property value specified by an SCA
2946
2947
        component definition into an instance of the Java type as defined by the XML to Java mapping in the
2948
         JAXB specification [JAXB] with XML schema validation enabled. [JCA90061]
2949
        When the Java type of a field, setter method or constructor parameter with the @Property annotation is
2950
        not a JAXB annotated class, the SCA runtime can use any XML to Java mapping when converting
2951
        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]
2952
2953
        Where there is both a setter method and a field for a property, the setter method is used.
                                                                                                                       Deleted: following
2954
        The @Property annotation has the attributes:
2955
            name (0..1) – the name of the property. For a field annotation, the default is the name of the field of
                                                                                                                       Deleted: optional
2956
            the Java class. For a setter method annotation, the default is the JavaBeans property name
2957
            [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
2958
2959
            be present. [JCA90013]
2960
            required (0..1) – a boolean value which specifies whether injection of the property value is required
                                                                                                                       Deleted: optional
            or not, where true means injection is required and false means injection is not required. Defaults to
2961
2962
            true. For a @Property annotation applied to a constructor parameter, the required attribute MUST
2963
            NOT have the value false. [JCA90014]
2964
                                                                                                                       Deleted: ¶
2965
                                                                                                                       The following snippet
2966
            For a @Property annotation, if the type of the Java class field or the type of the input parameter of
2967
            the setter method or constructor is defined as an array or as any type that extends or implements
2968
            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
2969
2970
             @many MUST be set to false.[JCA90047]
            The following snippet shows the definition of a configuration property using the @Property
2971
2972
            annotation for a collection.
2973
        Snippet 10-13 shows a property field definition sample.
2974
2975
             @Property(name="currency", required=true)
2976
             protected String currency;
2977
2978
             The following snippet shows a property setter sample
2979
2980
             @Property(name="currency", required=true)
```

public void setCurrency(String theCurrency) {

2981

2982 2983 2984

```
2985
            the setter method or constructor is defined as an array or as any type that extends or implements
2986
            java.util.Collection, then the SCA runtime MUST introspect the component type of the
2987
2988
            implementation with a cycle to true, otherwise
2989
             many MUST be set to false. [JCA90047]
2990
            The following snippet shows the definition of a configuration property using the @Property
2991
            annotation for a collection.
2992
          Snippet 10-13: Use of @Property on a Field
2993
2994
        For a @Property annotation, if the type of the Java class field or the type of the input parameter of the
2995
        setter method or constructor is defined as an array or as any type that extends or implements
2996
        ava.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.
2997
2998
        [JCA90047]
2999
        Snippet 10-14 shows the definition of a configuration property using the @Property annotation for a
3000
        collection.
3001
3002
            private List<String> helloConfigurationProperty;
3003
3004
            @Property(required=true)
3005
            public void setHelloConfigurationProperty(List<String> property) {
3006
                       helloConfigurationProperty = property;
3007
```

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

10.23 @Qualifier

<u>Figure</u> 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

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]

Field Code Changed

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

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 77 of 126

10.24 @Reference

Figure 10-24 defines the @Reference 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.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

3052 3053 3054

3055

3056

3057

3058

3060

3061

3062

3063 3064

3065

3066 3067

3068

3069

3070 3071

3072

3073

3074 3075

3076 3077

3078

3079

3080

3081

3033

3034

3035 3036

3037 3038

3039

3040

3041

3042

3043

3044

3045

3046

3047 3048

3049

3050 3051

> 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]

Field Code Changed

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

3059 The @Reference annotation has the attributes: Deleted: following Deleted: optional

- 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]

Deleted: optional

Snippet 10-15 shows a reference field definition sample.

Deleted: ¶ The following snippet

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

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

Deleted: ¶ The following snippet

Snippet 10-16 shows a reference setter sample

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

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

Deleted: ¶

The following fragment from

Deleted: component implementation

sca-iavacaa-1.1-spec-cd04

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

Page 78 of 126

06 Feb 2010

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.

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. <u>Snippet 10-18</u> shows the component type for the component implementation fragment in <u>Snippet 10-17</u>.

Deleted: The following snippet

Deleted: above

Deleted: 200903

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/> elementation 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]

Field Code Changed

Field Code Changed

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

Deleted: The following fragment from a component implementation

<u>Snippet 10-19</u> shows a sample of a service reference definition using the @Reference annotation on a java.util.List. The name of the reference is "helloServices" and its type is HelloService. The clientMethod() calls the "hello" operation of all the services referenced by the helloServices reference. In this case, at least one HelloService needs to be present, so *required* is true.

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

```
3135
                                                                                                                            Deleted: ...¶
                         for (int index = 0; index < helloServices.size(); index++) {</pre>
3136
3137
                                 HelloService helloService =
3138
                         (HelloService)helloServices.get(index);
3139
                                 String result = helloService.hello("Hello World!");
3140
                         }
3141
3142
                                                                                                                            Deleted: ¶
3143
           Snippet 10-19: Use of @Reference with a List of ServiceReferences
                                                                                                                            The following snippet
3144
3145
3146
             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
3147
3148
             [JCA90023]
3149
3150
         Snippet 10-20 shows the XML representation of the component type reflected from for the former
3151
         component implementation fragment. There is no need to author this component type in this case since it
3152
         can be reflected from the Java class.
3153
              <?xml version="1.0" encoding="ASCII"?>
3154
3155
              <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
                                                                                                                            Deleted: 200903
3156
3157
                 <!-- Any services offered by the component would be listed here -->
3158
                 <reference name="helloServices" multiplicity="1..n">
3159
                         <interface.java interface="services.hello.HelloService"/>
3160
                 </reference>
3161
3162
             </componentType>
3163
3164
             An unwired reference with a multiplicity of 0..1 MUST be presented to the implementation code by
3165
             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
3166
3167
             [JCA90023]
3168
           Snippet 10-20: Implied componentType for Implementation in Snippet 10-19
3169
3170
          An unwired reference with a multiplicity of 0..1 MUST be presented to the implementation code by the
3171
          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]
3172
           10.24.1 Reinjection
3173
         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.
3174
                                                                                                                            Field Code Changed
3175
3176
         [JCA90024]
3177
         In order for reinjection to occur, the following MUST be true:
3178
             1. The component MUST NOT be STATELESS scoped.
                 The reference MUST use either field-based injection or setter injection. References that are
3179
             injected through constructor injection MUST NOT be changed.
3180
3181
         [JCA90025]
3182
         Setter injection allows for code in the setter method to perform processing in reaction to a change.
         lf a reference target changes and the reference is not reinjected, the reference MUST continue to work as
3183
                                                                                                                            Field Code Changed
         if the reference target was not changed. [JCA90026]
3184
```

3185 If an operation is called on a reference where the target of that reference has been undeployed, the SCA Field Code Changed 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 3186 Deleted: 3187 Field Code Changed throw a ServiceUnavailableException. [JCA90028] If the target service of the reference is changed, the 3188 3189 reference MUST either continue to work or throw an InvalidServiceException when it is invoked. Field Code Changed 3190 [JCA90029] If it doesn't work, the exception thrown will depend on the runtime and the cause of the 3191 3192 ServiceReference that has been obtained from a reference by ComponentContext.cast() correspond Field Code Changed 3193 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 3194 3195 vas not changed. [JCA90030] If the target of a ServiceReference has been undeployed, the SCA runtime Field Code Changed 3196 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 3197 Field Code Changed hrow a ServiceUnavailableException when an operation is invoked on the ServiceReference 3198 3199 [JCA90032] If the target service of a ServiceReference is changed, the reference MUST either continu Field Code Changed 3200 to work or throw an InvalidServiceException when it is invoked. [JCA90033] If it doesn't work, the 3201 exception thrown will depend on the runtime and the cause of the failure. 3202 A reference or ServiceReference accessed through the component context by calling getService() or Field Code Changed 3203 getServiceReference() MUST correspond to the current configuration of the domain. This applies whether 3204 or not reinjection has taken place. [JCA90034] <mark>If the target of a reference or ServiceReference accessed</mark> through the component context by calling getService() or getServiceReference() has been undeployed or 3205 3206 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 3207 ServiceUnavailableException, [JCA90035] If the target service of a reference or ServiceReference 3208 Deleted: 3209 accessed through the component context by calling getService() or getServiceReference() has changed, 3210 the returned value SHOULD be a reference to the changed service. [JCA90036] 3211 The rules for reference reinjection also apply to references with a multiplicity of 0..n or 1..n. This means 3212 that in the cases where reference reinjection is not allowed, the array or Collection for a reference of 3213 multiplicity 0..n or multiplicity 1..n MUST NOT change its contents when changes occur to the reference 3214 wiring or to the targets of the wiring. [JCA90037] In cases where the contents of a reference array or Deleted: 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. 3215 3216 [JCA90038] A reinjected array or Collection for a reference MUST NOT be the same array or Collection 3217 Deleted: 3218 bject previously injected to the component. [JCA90039] 3219

	Effect on			
Change event	Injected Reference or ServiceReference	Existing ServiceReference Object**	Subsequent invocations of ComponentContext.getService Reference() 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	Business methods throw ServiceUnavailableExce	Business methods throw ServiceUnavailableExce	Result is be a reference to the unavailable service. Business methods throw	

unavailable	ption	ption	ServiceUnavailableException.
Target service changed	might continue to work, depending on the runtime and the type of change that was made. If it doesn't work, the exception thrown will depend on the runtime and the cause of the failure.	might continue to work, depending on the runtime and the type of change that was made. If it doesn't work, the exception thrown will depend on the runtime and the cause of the failure.	Result is a reference to the changed service.

^{*} Other conditions:

The component cannot be STATELESS scoped.

The reference has to use either field-based injection or setter injection. References that are injected through constructor injection cannot be changed.

** Result of invoking ComponentContext.cast() corresponds to the reference that is passed as a parameter to cast().

Table 10-1Reinjection Effects

10.25 @Remotable

<u>Figure</u> 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

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]

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.

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

```
Deleted: annotate a Java
```

Deleted: The following snippet

Deleted: or to annotate a Java class (used to define an interface) as

Field Code Changed

package services.hello;

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 82 of 126

```
import org.oasisopen.sca.annotation.*;

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

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

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

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

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

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

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

Deleted: The following snippet

Deleted: remotable

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

@Service(HelloService.class)

@Remotable

import org.oasisopen.sca.annotation.*;

public String hello(String message) {

public class HelloServiceImpl implements HelloService {

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

```
package services.hello;
import org.oasisopen.sca.annotation.*;
public interface HelloService {
```

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 83 of 126

```
3307
3308
              String hello(String message);
3309
3310
3311
           package services.hello;
3312
3313
           import org.oasisopen.sca.annotation.*;
3314
3315
           public class HelloClient {
3316
3317
              @Remotable
3318
              @Reference
              protected HelloService myHello;
3319
3320
3321
              public String greeting(String message) {
3322
                     return myHello.hello(message);
3323
3324
3325
```

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

10.26 @Requires

3326

3327

3328 3329

3330 3331

3332

3333

3334

3335

3336 3337

3338

3339

3340 3341

3342

3343

3344

3345 3346

3347 3348

3349 3350

3351 3352

3353

3354 3355

3356

3357 3358

3359

Figure 10-26 defines the @Requires 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.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

The @Scope annotation MUST only be used on a service's implementation class. It is an error to use this annotation on an interface.Figure 10-27 defines the @Scope annotation:

Deleted: The following Java code

sca-javacaa-1.1-spec-cd04 Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 84 of 126

```
3360
3361
          package org.oasisopen.sca.annotation;
3362
3363
          import static java.lang.annotation.ElementType.TYPE;
3364
          import static java.lang.annotation.RetentionPolicy.RUNTIME;
3365
          import java.lang.annotation.Retention;
3366
          import java.lang.annotation.Target;
3367
3368
          @Target(TYPE)
          @Retention(RUNTIME)
3369
3370
          public @interface Scope {
3371
3372
             String value() default "STATELESS";
3373
        The @Scope annotation MUST only be used on a service's implementation class. It is an error to use this
3374
3375
       annotation on an interface. Figure 10-27: Scope Annotation
3376
3377
      The @Scope annotation MUST only be used on a service's implementation class. It is an error to use this
3378
       annotation on an interface. [JCA90041]
      The @Scope annotation has the attribute:
                                                                                              Deleted: following
3379
                                                                                              Deleted:
3380
          value – the name of the scope
3381
          SCA defines the following scope names, but others can be defined by particular Java-based
                                                                                              Deleted: :
          implementation types,
3382
                                                                                              Deleted:
3383
             STATELESS,_____
                                                                                              Deleted:
             COMPOSITE
3384
3385
         The default value is STATELESS.
                                                                                              Deleted: The following snippet
      Snippet 10-24 shows a sample for a COMPOSITE scoped service implementation:
3386
3387
3388
          package services.hello;
3389
3390
          import org.oasisopen.sca.annotation.*;
3391
3392
          @Service(HelloService.class)
3393
          @Scope("COMPOSITE")
3394
          public class HelloServiceImpl implements HelloService {
3395
3396
             public String hello(String message) {
3397
3398
3399
3400
        Snippet 10-24: Use of @Scope
        10.28 @Service
3401
                                                                                              Deleted: The following Java code
3402
      Figure 10-28 defines the @Service annotation:
3403
3404
          package org.oasisopen.sca.annotation;
3405
3406
          import static java.lang.annotation.ElementType.TYPE;
3407
          import static java.lang.annotation.RetentionPolicy.RUNTIME;
3408
          import java.lang.annotation.Retention;
3409
          import java.lang.annotation.Target;
```

06 Feb 2010 Page 85 of 126

sca-javacaa-1.1-spec-cd04 Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
@Target(TYPE)
@Retention(RUNTIME)
public @interface Service {
   Class<?>[] value();
   String[] names() default {};
```

Figure 10-28: Service Annotation

3410 3411

3412 3413

3414 3415

3416

3417

3418

3419 3420

3421

3422

3423 3424

3425 3426

3427

3428

3429

3430

3431

3432

3433 3434

3435

3436

3437

3438

3439

3440

3441

3442

3443

3444

3445

3446

3447

3448

3449

3450

3451 3452

3453

3454 3455 3456 The @Service annotation is used on a component implementation class to specify the SCA services offered by the implementation. An implementation class need not be declared as implementing all of the interfaces implied by the services declared in its @Service annotation, but all methods of all the declared service interfaces MUST be present. [JCA90042] A class used as the implementation of a service is not required to have a @Service annotation. If a class has no @Service annotation, then the rules determining which services are offered and what interfaces those services have are determined by the specific implementation type.

The @Service annotation has the attributes:

- **value**(1..1) An array of interface or class objects that are exposed as services by this implementation. If the array is empty, no services are exposed.
 - name (0..1) A string which is used as the service name. [JCA90048]

names (0..1) - An array of Strings which are used as the service names for each of the interfaces declared in the value array. [JCA90049] The number of Strings in the names attribute array of the @Service annotation MUST match the number of elements in the value attribute array. [JCA90050], The value of each element in the @Service names array MUST be unique amongst all the other element values in the array. [JCA90043]

[JCA90044] [JCA90051 [JCA90060]

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

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

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

```
package services.hello;
import org.oasisopen.sca.annotation.Service;
@Service(HelloService.class)
public class HelloServiceImpl implements HelloService {
    public void hello(String name) {
        System.out.println("Hello " + name);
```

Snippet 10-25: Use of @Service

Deleted:

Deleted: interfaces() default Void.class };¶

Deleted:

String name() default "";¶

Deleted: .Class<?> value() default Void class;

Deleted: }¶

Field Code Changed

Deleted: following

Deleted: interfaces (1..1) - The

Deleted: is an

Deleted: Contains an

Deleted: interfaces

Deleted:

Deleted: ¶

<#>value - A shortcut for the case when the class provides only a single service interface contains a single interface or class object that is exposed as a service by this component implementation. ¶

Field Code Changed

Deleted: names of the defined services default

Deleted: s

Deleted: the interfaces

Deleted: parameter

Deleted: .

Deleted: service name for each interface in the interfaces

Field Code Changed

Deleted: The following snippet

11 WSDL to Java and Java to WSDL

3457 3458

3459 3460

3461

3462 3463

3464 3465

3466 3467

3468

3469

3470

3471

3472

3473

3474 3475

3476 3477

3478

3479

[JCA100003]

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.

definition instead of the annotated Java class.

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.

Impact to SCA Interface Annotation **Property** A Java interface or class annotated with @WebService @WebService MUST be treated as if annotated with the SCA @Remotable annotation [JCA100012] If used to define a service, sets service name name targetNamespace <u>None</u> serviceName None A Java class annotated with the @WebService wsdlLocation annotation with its wsdlLocation attribute set MUST have its interface defined by the referenced WSDL

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010

Field Code Changed

Field Code Changed

Page 87 of 126

[JCA100013]

A Java class annotated with the @WebService endpointInterface

annotation with its endpointInterface attribute set MUST

have its interface defined by the referenced interface

instead of annotated Java class. [JCA100014]

portName **None**

@WebMethod

Sets operation name operationName

action **None**

<u>exclude</u> Method is excluded from the interface.

The SCA runtime MUST treat an @OneWay

@org.oasisopen.sca.annotation.OneWay annotation as

a synonym for the @javax.jws.OneWay annotation.

[JCA100002]

@WebParam

Sets parameter name <u>name</u>

targetNamespace **None**

mode Sets directionality of parameter

<u>header</u> 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**

<u>header</u> 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

<u>parameterStyle</u>

@HandlerChain None

<u>file</u>

<u>name</u>

Table 11-1: JSR 181 Annotations and SCA Interfaces

3480

3481

<u>Annotation</u> **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]

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 89 of 126

Annotation Impact to SCA Interface **Property** <u>value</u> @WebFault Sets fault name <u>name</u> targetNamespace **None** <u>faultBean</u> None @RequestWrapper **None** <u>localName</u> targetNamespace <u>className</u> @ResponseWrapper <u>None</u> <u>localName</u> targetNamespace <u>className</u> @WebServiceClient An interface or class annotated with @WebServiceClient MUST NOT be used to define an SCA interface. [JCA100018] <u>name</u> targetNamespace wsdlLocation @WebEndpoint **None**

Annotation Impact to SCA Interface **Property** <u>name</u> @WebServiceProvider A class annotated with @WebServiceProvider MUST be treated as if annotated with the SCA @Remotable annotation. [JCA100019] A Java class annotated with the @WebServiceProvider **wsdlLocation** 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 <u>None</u> <u>value</u> @WebServiceRef See JEE specification <u>name</u> wsdlLocation type <u>value</u> mappedName

See JEE specification

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

@WebServiceRefs

06 Feb 2010 Page 91 of 126

Annotation Impact to SCA Interface **Property** value @Action **None** fault input output @FaultAction None value output

Table 11-2: JSR 224 Annotations and SCA Interfaces

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.

or SCA service interfaces defined using interface java, the Java interface MUST NOT contain the <u>additional client-side asynchronous polling and callback methods defined by JAX-WS.</u> [JCA100006] <mark>For</mark> SCA reference interfaces defined using interface.java, the SCA runtime MUST support a Java interface 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

Field Code Changed

3482

3483 3484

3485

3486

3487

3488 3489

3490

3491

3492

3493

3494 3495

3496

3497

3498

3499

3500

3501

3502

3503 3504

3505

3506

3507

3508

mplementation. [JCA100008] The additional client-side asynchronous polling and callback methods defined by JAX-WS are recognized Field Code Changed

Deleted: ¶

in a Java interface according to the steps: 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
- 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<?>

sca-javacaa-1.1-spec-cd04 Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 92 of 126

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, if

11.3 Treatment of SCA Asynchronous Service API

For SCA service interfaces defined using interface.java, the SCA runtime MUST support a Java interface hich contains the server-side asynchronous methods defined by SCA. [JCA100010]

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

sca-javacaa-1.1-spec-cd04 Copyright © OASIS® 2005, 2010. All Rights Reserved.

3550

3551

3552

3553

3554 3555

3556 3557

> 06 Feb 2010 Page 93 of 126

12 Conformance

The XML schema pointed to by the RDDL document at the namespace URI, defined by this specification, 3561 3562 are considered to be authoritative and take precedence over the XML schema defined in the appendix of 3563 this document.

3564 Normative code artifacts related to this specification are considered to be authoritative and take. precedence over specification text. 3565

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

- a) SCA Java XML Document,
- b) SCA Java Class
- c) SCA Runtime.
- 3569

3560

3566

3567

3568

3570

3576

3581 3582

3583 3584

3585

3586 3587

3588 3589

3590 3591

3592

12.1 SCA Java XML Document

An SCA Java XML document is an SCA Composite Document, or an SCA ComponentType Document, 3571 as defined by the SCA Assembly Model specification [ASSEMBLY], that uses the <interface.java> 3572 element. Such an SCA Java XML document MUST be a conformant SCA Composite Document or SCA 3573

ComponentType Document, as defined by the SCA Assembly Model specification [ASSEMBLY], and 3574

MUST comply with the requirements specified in the Interface section of this specification. 3575

Deleted: or an SCA ConstrainingType Document

Deleted: the code artifacts

Deleted: For

Deleted: s

06 Feb 2010

Page 95 of 126

Deleted: or SCA ConstrainingType Document

Deleted: , the specification text is

12.2 SCA Java Class

3577 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 3578 3579 and APIs defined in this specification MUST comply with the requirements specified in this specification for those annotations and APIs. 3580

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:

- The implementation MUST meet all the conformance requirements defined by the SCA Assembly Model Specification [ASSEMBLY].
- The implementation MUST support <interface.java> and MUST comply with all the normative statements in Section 3.
- 3593 The implementation MUST reject an SCA Java XML Document that does not conform to the sca-3594 interface-java.xsd schema.
- 3595 The implementation MUST support and comply with all the normative statements in Section 10.

A. XML Schema: sca-interface-java-1.1.xsd

3596

```
3597
            <?xml version="1.0" encoding="UTF-8"?>
3598
            <!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved.
                                                                                                              Deleted: 2009
            OASIS trademark, IPR and other policies apply. <schema xmlns="http://www.w3.org/2001/XMLSchema"
3599
3600
3601
               targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
                                                                                                              Deleted: 200903
3602
               xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3603
               elementFormDefault="qualified">
                                                                                                              Deleted: 200903
3604
3605
               <include schemaLocation="sca-core-1.1-cd05.xsd"/>
                                                                                                              Deleted: cd03
3606
3607
               <!-- Java Interface -->
3608
               <element name="interface.java" type="sca:JavaInterface"</pre>
3609
                         substitutionGroup="sca:interface"/>
3610
               <complexType name="JavaInterface">
3611
                   <complexContent>
3612
                      <extension base="sca:Interface">
3613
                         <sequence>
3614
                            <any namespace="##other" processContents="lax" minOccurs="0"</pre>
3615
                                 maxOccurs="unbounded"/>
3616
                         </sequence>
3617
                         <attribute name="interface" type="NCName" use="required"/>
3618
                         <attribute name="callbackInterface" type="NCName"
3619
                                     use="optional"/>
3620
                      </extension>
                                                                                                              Deleted:
3621
                   </complexContent>
3622
               </complexType>
                                                                                                              namespace="##other"
3623
                                                                                                              processContents="lax"/>¶
3624
            </schema>
```

sca-javacaa-1.1-spec-cd04 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;
    \frac{/**}{*} 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() {
```

```
3681
                * Constructor used by concrete subclasses
3682
                * @param domainURI - The Domain URI of the Domain accessed via this
3683
                * SCAClientFactory
3684
3685
               protected SCAClientFactory(URI domainURI) {
3686
              throws NoSuchDomainException {
3687
                   this.domainURI = domainURI;
3688
3689
3690
3691
                * Gets the Domain URI of the Domain accessed via this SCAClientFactory
3692
                  @return - the URI for the Domain
3693
3694
               protected URI getDomainURI() {
3695
                   return domainURI;
3696
3697
3698
3699
3700
                * Creates a new instance of the SCAClient that can be
3701
                * used to lookup SCA Services.
3702
3703
                * @param domainURI
                                        URI of the target domain for the SCAClient
3704
                * @return A new SCAClient
3705
3706
               public static SCAClientFactory newInstance( URI domainURI )
3707
                    throws NoSuchDomainException {
3708
                   return newInstance(null, null, domainURI);
3709
3710
3711
3712
                * Creates a new instance of the SCAClient that can be
3713
                  used to lookup SCA Services.
3714
3715
                * @param properties Properties that may be used when
3716
                * creating a new instance of the SCAClient
3717
                * @param domainURI
                                       URI of the target domain for the SCAClient
3718
                  @return A new SCAClient instance
3719
3720
               public static SCAClientFactory newInstance(Properties properties,
3721
                                                                           URI domainURI)
3722
                     throws NoSuchDomainException {
3723
                   return newInstance(properties, null, domainURI);
3724
3725
3726
3727
                * Creates a new instance of the SCAClient that can be
3728
                * used to lookup SCA Services.
3729
3730
                * @param classLoader
                                      ClassLoader that may be used when
3731
                  creating a new instance of the SCAClient
                * @param domainURI
3732
                                       URI of the target domain for the SCAClient
3733
                * @return A new SCAClient instance
3734
3735
               public static SCAClientFactory newInstance(ClassLoader classLoader,
3736
                                                                           URI domainURI)
3737
                     throws NoSuchDomainException {
3738
                   return newInstance(null, classLoader, domainURI);
3739
3740
3741
3742
                * Creates a new instance of the SCAClient that can be
3743
                * used to lookup SCA Services.
3744
```

```
3745
                                       Properties that may be used when
                 * @param properties
3746
                * creating a new instance of the SCAClient
3747
                * @param classLoader ClassLoader that may be used when
3748
                  creating a new instance of the SCAClient
3749
                  @param domainURI
                                        URI of the target domain for the SCAClient
3750
                * @return A new SCAClient instance
3751
3752
               public static SCAClientFactory newInstance(Properties properties,
3753
                                                        ClassLoader classLoader,
3754
                                                        URI domainURI)
3755
                     throws NoSuchDomainException {
3756
                   final SCAClientFactoryFinder finder
3757
                       factoryFinder != null ? factoryFinder :
3758
                           new SCAClientFactoryFinderImpl();
3759
                   final SCAClientFactory factory
                       = finder.find(properties, classLoader, domainURI);
3760
3761
                   return factory;
3762
3763
3764
                * Returns a reference proxy that implements the business interface <T>
3765
3766
                * of a service in the SCA Domain handled by this SCAClientFactory
3767
3768
                * @param serviceURI the relative URI of the target service. Takes the
3769
                  form componentName/serviceName.
3770
                * Can also take the extended form componentName/serviceName/bindingName
3771
                * to use a specific binding of the target service
3772
3773
                * @param interfaze The business interface class of the service in the
3774
                  domain
                * @param <T> The business interface class of the service in the domain
3775
3776
3777
                  @return a proxy to the target service, in the specified SCA Domain
3778
                * that implements the business interface <B>.
3779
                  @throws NoSuchServiceException Service requested was not found
3780
                * @throws NoSuchDomainException Domain requested was not found
3781
3782
               public abstract <T> T getService(Class<T> interfaze, String serviceURI)
3783
                   throws NoSuchServiceException, NoSuchDomainException;
3784
```

B.1.2 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.

```
* Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.

* OASIS trademark, IPR and other policies apply.

*/

package org.oasisopen.sca.client;

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

import org.oasisopen.sca.NoSuchDomainException;

/* A Service Provider Interface representing a SCAClientFactory finder.

* SCA provides a default reference implementation of this interface.

* SCA runtime vendors can create alternative implementations of this

* interface that use different class loading or lookup mechanisms.
```

```
3806
3807
              public interface SCAClientFactoryFinder {
3808
3809
                  * Method for finding the SCAClientFactory for a given Domain URI using
3810
3811
                  * 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
3812
3813
3814
                  * @param domainURI - the Domain URI - must be a valid SCA Domain URI
* @return - the SCAClientFactory or null if the factory could not be
3815
3816
                  * @throws - NoSuchDomainException if the domainURI does not reference
3817
                     a valid SCA Domain
                  * found
3818
3819
3820
                   SCAClientFactory find(Properties properties,
3821
                                              ClassLoader classLoader,
                                              URI domainURI )
3822
3823
                         throws NoSuchDomainException ;
3824
```

B.1.3 SCAClientFactoryFinderImpl class

3825 3826

3827 3828

3829

3830

3831 3832

3833

3834

3835

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

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

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

```
3836
            /*
  * Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
3837
             * OASIS trademark, IPR and other policies apply.
3838
3839
3840
            package org.oasisopen.sca.client.impl;
3841
3842
            import org.oasisopen.sca.client.SCAClientFactoryFinder;
3843
3844
            import java.io.BufferedReader;
3845
            import java.io.Closeable;
3846
            import java.io.IOException;
3847
            import java.io.InputStream;
3848
            import java.io.InputStreamReader;
3849
            import java.lang.reflect.Constructor;
3850
            import
                   java.net.URI;
3851
            import java.net.URL;
3852
            import java.util.Properties;
3853
3854
            import org.oasisopen.sca.NoSuchDomainException;
3855
            import org.oasisopen.sca.ServiceRuntimeException;
3856
            import org.oasisopen.sca.client.SCAClientFactory;
3857
3858
3859
            * This is a default implementation of an SCAClientFactoryFinder which is
3860
             * used to find an implementation of the SCAClientFactory interface.
3861
3862
           * @see SCAClientFactoryFinder
* @see SCAClientFactory
3863
3864
```

```
3865
            * @author OASIS Open
3866
3867
           public class SCAClientFactoryFinderImpl implements SCAClientFactoryFinder {
3868
3869
3870
                * The name of the System Property used to determine the SPI
3871
                * implementation to use for the SCAClientFactory.
3872
3873
               private static final String SCA_CLIENT_FACTORY_PROVIDER_KEY =
3874
                    SCAClientFactory.class.getName();
3875
3876
3877
                * The name of the file loaded from the ClassPath to determine
3878
                * the SPI implementation to use for the SCAClientFactory.
3879
               3880
3881
3882
3883
                /**
                * Public Constructor
3884
3885
                * /
3886
               public SCAClientFactoryFinderImpl() {
3887
3888
3889
3890
                * Creates an instance of the SCAClientFactorySPI implementation.
3891
                * This discovers the SCAClientFactorySPI Implementation and instantiates
                * the provider's implementation.
3892
3893
                * @param properties
3894
                                       Properties that may be used when creating a new
3895
                * instance of the SCAClient
3896
                * @param classLoader ClassLoader that may be used when creating a new
3897
                  instance of the SCAClient
3898
                * @return new instance of the SCAClientFactory
3899
                  @throws ServiceRuntimeException Failed to create SCAClientFactory
                * Implementation.
3900
3901
                * /
3902
               public SCAClientFactory find(Properties properties,
3903
                                            ClassLoader classLoader,
3904
                                             URI domainURI )
3905
                     throws NoSuchDomainException, ServiceRuntimeException {
3906
                     if (classLoader == null) {
3907
                           classLoader = getThreadContextClassLoader ();
3908
3909
                     final String factoryImplClassName =
                     discoverProviderFactoryImplClass(properties, classLoader);
final Class<? extends SCAClientFactory> factoryImplClass
3910
3911
3912
                           = loadProviderFactoryClass(factoryImplClassName,
3913
                                                            classLoader);
3914
                     final SCAClientFactory factory =
3915
                            instantiateSCAClientFactoryClass(factoryImplClass,
3916
                                                                domainURI );
3917
                     return factory;
3918
3919
3920
3921
                * Gets the Context ClassLoader for the current Thread.
3922
3923
                  @return The Context ClassLoader for the current Thread.
3924
3925
               private static ClassLoader getThreadContextClassLoader () {
3926
                   final ClassLoader threadClassLoader =
3927
                    Thread.currentThread().getContextClassLoader();
3928
                   return threadClassLoader;
```

```
3929
3930
3931
3932
                 * Attempts to discover the class name for the SCAClientFactorySPI
3933
                   implementation from the specified Properties, the System Properties
3934
                 * or the specified ClassLoader.
3935
3936
                 * @return The class name of the SCAClientFactorySPI implementation
3937
                  @throw ServiceRuntimeException Failed to find implementation for
                 * SCAClientFactorySPI.
3938
3939
3940
                private static String
3941
                    discoverProviderFactoryImplClass(Properties properties,
3942
                                                           ClassLoader classLoader)
3943
                    throws ServiceRuntimeException {
3944
                    String providerClassName =
3945
                     checkPropertiesForSPIClassName(properties);
3946
                    if (providerClassName != null) {
3947
                        return providerClassName;
3948
3949
3950
                    providerClassName =
3951
                     checkPropertiesForSPIClassName(System.getProperties());
3952
                    if (providerClassName != null) {
3953
                        return providerClassName;
3954
3955
3956
                    providerClassName = checkMETAINFServicesForSIPClassName(classLoader);
3957
                    if (providerClassName == null) {
3958
                        throw new ServiceRuntimeException(
3959
                             "Failed to find implementation for SCAClientFactory");
3960
3961
3962
                    return providerClassName;
3963
3964
3965
3966
                 * Attempts to find the class name for the SCAClientFactorySPI
3967
                  implementation from the specified Properties.
3968
3969
                  @return The class name for the SCAClientFactorySPI implementation
3970
                 * or <code>null</code> if not found.
3971
3972
                private static String
3973
                    checkPropertiesForSPIClassName(Properties properties) {
3974
                    if (properties == null) {
3975
                        return null;
3976
3977
3978
                    final String providerClassName =
                    properties.getProperty(SCA_CLIENT_FACTORY_PROVIDER_KEY);
if (providerClassName != null && providerClassName.length() > 0) {
3979
3980
3981
                        return providerClassName;
3982
3983
3984
                    return null;
3985
3986
3987
3988
                 * Attempts to find the class name for the SCAClientFactorySPI
3989
                  implementation from the META-INF/services directory
3990
3991
                 * @return The class name for the SCAClientFactorySPI implementation or
3992
                   <code>null</code> if not found.
```

```
3993
3994
                private static String checkMETAINFServicesForSIPClassName(ClassLoader cl)
3995
3996
                    final URL url =
3997
                     cl.getResource(SCA_CLIENT_FACTORY_PROVIDER_META_INF_SERVICE);
3998
                    if (url == null) {
3999
                        return null;
4000
4001
4002
                    InputStream in = null;
4003
                    try {
4004
                        in = url.openStream();
4005
                        BufferedReader reader = null;
4006
                        try {
4007
                            reader =
4008
                                  new BufferedReader(new InputStreamReader(in, "UTF-8"));
4009
4010
                            String line;
4011
                            while ((line = readNextLine(reader)) != null) {
4012
                                if (!line.startsWith("#") && line.length() > 0) {
4013
                                    return line;
4014
4015
4016
4017
                            return null;
4018
                         finally {
4019
                            closeStream(reader);
4020
4021
                    } catch (IOException ex) {
4022
                        throw new ServiceRuntimeException(
4023
                                   "Failed to discover SCAClientFactory provider", ex);
4024
                      finally {
4025
                        closeStream(in);
4026
4027
4028
4029
4030
                 * Reads the next line from the reader and returns the trimmed version
4031
                 * of that line
4032
4033
                 * @param reader The reader from which to read the next line
4034
                 * @return The trimmed next line or <code>null</code> if the end of the
4035
                  stream has been reached
4036
                 * @throws IOException I/O error occurred while reading from Reader
4037
4038
                private static String readNextLine(BufferedReader reader)
4039
                    throws IOException {
4040
4041
                    String line = reader.readLine();
4042
                    if (line != null) {
4043
                        line = line.trim();
4044
4045
                    return line;
4046
4047
4048
4049
                 ^{\star} Loads the specified SCAClientFactory Implementation class.
4050
4051
                   @param factoryImplClassName The name of the SCAClientFactory
4052
                 * Implementation class to load
4053
                 * @return The specified SCAClientFactory Implementation class
4054
                 * @throws ServiceRuntimeException Failed to load the SCAClientFactory
4055
                 * Implementation class
4056
```

```
4057
                private static Class<? extends SCAClientFactory>
4058
                    load Provider Factory Class (String\ factory Impl Class Name,
4059
                                              ClassLoader classLoader)
4060
                    throws ServiceRuntimeException {
4061
4062
4063
                        final Class<?> providerClass =
4064
                            classLoader.loadClass(factoryImplClassName);
4065
                        final Class<? extends SCAClientFactory> providerFactoryClass =
4066
                            providerClass.asSubclass(SCAClientFactory.class);
4067
                        return providerFactoryClass;
4068
                      catch (ClassNotFoundException ex)
4069
                        throw new ServiceRuntimeException(
4070
                             "Failed to load SCAClientFactory implementation class "
4071
                             + factoryImplClassName, ex);
4072
                     catch (ClassCastException ex) {
4073
                        throw new ServiceRuntimeException(
4074
                                   "Loaded SCAClientFactory implementation class "
4075
                                   + factoryImplClassName
4076
                                     is not a subclass of
4077
                                 + SCAClientFactory.class.getName() , ex);
4078
4079
4080
4081
4082
                 * Instantiate an instance of the specified SCAClientFactorySPI
4083
                 * Implementation class.
4084
4085
                 * @param factoryImplClass The SCAClientFactorySPI Implementation
                 * class to instantiate.
* @return An instance of the SCAClientFactorySPI Implementation class
4086
4087
4088
                 * @throws ServiceRuntimeException Failed to instantiate the specified
4089
                   specified SCAClientFactorySPI Implementation class
4090
4091
                private static SCAClientFactory instantiateSCAClientFactoryClass(
4092
                                   Class<? extends SCAClientFactory> factoryImplClass,
4093
                            URI domainURI)
4094
                    throws NoSuchDomainException, ServiceRuntimeException {
4095
4096
4097
                        Constructor<? extends SCAClientFactory> URIConstructor =
4098
                            factoryImplClass.getConstructor(domainURI.getClass());
4099
                        SCAClientFactory provider =
4100
                           URIConstructor.newInstance( domainURI );
4101
                        return provider;
4102
                      catch (Throwable ex) {
4103
                        throw new ServiceRuntimeException(
                           "Failed to instantiate SCAClientFactory implementation class "
4104
4105
                           + factoryImplClass, ex);
4106
4107
4108
4109
4110
                 * Utility method for closing Closeable Object.
4111
4112
                  @param closeable The Object to close.
4113
4114
                private static void closeStream(Closeable closeable) {
4115
                    if (closeable != null) {
4116
                        try{
4117
                            closeable.close();
4118
                          catch (IOException ex) {
4119
                            throw new ServiceRuntimeException("Failed to close stream",
4120
```

4121 4122 4123 4124	<u>}</u>
4125	B.1.4 SCAClient Classes and Interfaces - what does a vendor need to do?
4126 4127 4128	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.
4129	Implement their SCAClientFactory implementation class
4130 4131 4132 4133	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).
4134	Configure the Vendor SCAClientFactory implementation class so that it gets used
4135	Vendors have several options:
4136	Option 1: Set System Property to point to the Vendor's implementation
4137 4138	Vendors set the org.oasisopen.sca.client.SCAClientFactory System Property to point to their implementation class and use the reference implementation of SCAClientFactoryFinder
4139	Option 2: Provide a META-INF/services file
4140 4141	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
4142 4143	Option 3: Inject a vendor implementation of the SCAClientFactoryFinder interface into SCAClientFactory
4144 4145 4146 4147 4148	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.

C. Conformance Items

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

4151 4152

4149

4150

Conformance ID	Description	
[JCA20001]	Remotable Services MUST NOT make use of method overloading.	Field Code Changed
[JCA20002]	the SCA runtime MUST ensure that a stateless scoped implementation instance object is only ever dispatched on one thread at any one time.	Field Code Changed
[JCA20003]	within the SCA lifecycle of a stateless scoped implementation instance, the SCA runtime MUST only make a single invocation of one business method.	Field Code Changed
[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.	Field Code Changed
[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.	Field Code Changed
[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.	Field Code Changed
[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.	Field Code Changed
[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.	Field Code Changed
[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".	Field Code Changed
[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".	Field Code Changed
[JCA30001]	The value of the @interface attribute MUST be the fully qualified name of the Java interface class	Field Code Changed
[JCA30002]	The value of the @callbackInterface attribute MUST be the fully	Field Code Changed

	qualified name of a Java interface used for callbacks		
	<u> </u>		Field Code Changed
[JCA30003]	if the Java interface class identified by the @interface attribute does contain a Java @Callback annotation, then the Java interface class	-/	Field Code Changed
	identified by the @callbackInterface attribute MUST be the same	/	Field Code Changed
	interface class.	-//	Field Code Changed
[JCA30004]	The interface java element MUST conform to the schema defined in	11	Deleted: implementation class
29191904	the sca-interface-java.xsd schema.	_ /	Deleted: or the @callbackInterface
[JCA30005]	The value of the @remotable attribute on the <interface.java></interface.java> element	1 11	Deleted: any of
	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.		Deleted: @AllowsPassByReference, @ComponentName, @Constructor, @Context, @Destroy, @EagerInit, @Init, Deleted: Property, @
1		M/J	Deleted: , @Reference, @Scope,
[JCA30006]	A Java interface referenced by the @interface attribute of an <interface.java></interface.java> element MUST NOT contain the following SCA Java	1/ / //	@Service
1	annotations:		Field Code Changed
1	@Intent, @Qualifier,	111	Deleted: callbackInterface
110400007		11/	Deleted: Callback, @
[JCA30007]	A Java interface referenced by the @interface, attribute of an <interface.java></interface.java> element MUST NOT contain any of the following SCA	." /	Field Code Changed
	Java annotations:	1/	Deleted: A Java interface referenced by the @callbackInterface attribute of
L	@AllowsPassByReference, @ComponentName, @Constructor, @Context, @Destroy, @EagerInit, @Init, @Intent, @Property, @Qualifier, @Reference, @Scope, @Service.		an <interface.java></interface.java> element MUST NOT contain any of the following SCA Java annotations:¶ @AllowsPassByReference, @Callback, @ComponentName,
[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	<u>.</u> /	@Constructor, @Context, @Destroy, @EagerInit, @Init, @Intent, @Property, @Qualifier, @Reference, @Scope, @Service.
	every method that is present in both interfaces is defined consistently in both interfaces with respect to the @OneWay annotation, that is, the	,	Field Code Changed
	annotation is either present in both interfaces or absent in both interfaces.	11	Deleted: The SCA Runtime MUST call a constructor of the component implementation at the start of the Constructing state.
[JCA30010]	If the identified class is annotated with either the JAX-WS		Field Code Changed
	@WebService or @WebServiceProvider annotations and the annotation has a non-empty wsdlLocation property, then the SCA	,	Deleted: perform any construc [1]
	Runtime MUST act as if an <interface.wsdl></interface.wsdl> element is present	1,	Deleted: the
	instead of the <interface.java></interface.java> element, with an @interface attribute	11	Deleted: at the start of the [2]
	identifying the portType mapped from the Java interface class and containing @requires and @policySets attribute values equal to the		Field Code Changed
	@requires and @policySets attribute values of the <interface.java></interface.java>		Deleted: The SCA Runtime M [3]
	element.	W///	Field Code Changed
[JCA40001]	The SCA Runtime MUST call a constructor of a component	M/M	Deleted: If an exception is through
	implementation,		Deleted: constructor complete [5]
[JCA40002]	When the constructor completes successfully, the SCA Runtime MUST	11 11	Deleted: Terminated
	transition the component implementation to the Injecting state.		Field Code Changed
[JCA40003]	When the Constructing state, the SCA Runtime MUST transition the	1///	Deleted: When a component [6]
	component implementation to the Injecting state.		Deleted: thrown whilst
[JCA40004]	Jf an exception is in the Injecting state, the SCA Runtime MUST	11/-	Deleted: Constructing
	transition the component implementation.	- 	Deleted: first inject all field and [7]
			Deleted: to the Terminated state

[JCA40005]	When a component implementation instance is in the Injecting state, the SCA Runtime MUST inject all field and setter properties that are	Field Code Changed Deleted: firstreferences, after
	present into the component implementation.	all the properties have been inj [8]
[JCA40006]	When a component implementation instance is in the Injecting state,	Field Code Changed
100/1400001	the SCA Runtime MUST inject all field and setter references that are	Deleted: The SCA Runtime MUST
	present into the component implementation, after all the properties	ensure that the correct synchronization model is used so that
	have been injected.	all injected properties and references
[JCA40007]	The SCA Runtime MUST NOT invoke Service methods on the component	are made visible to the component implementation without requiring the
	implementation when the component implementation is in the Injecting	component implementation developer to do any specific synchronization.
	state.	Field Code Changed
[JCA40008]	The SCA Runtime MUST transition the component implementation to	Deleted: ensure that the correct
	the Initializing, state.	synchronization model is used so that
[JCA40009]	When the injection of properties or references, the SCA Runtime MUST	all injected properties and references are made visible towithout
	transition the component implementation to the Destroying state.	requiringdeveloper to do any [9]
[JCA40010]	If an exception is thrown whilst injecting properties or references, the	Field Code Changed
	SCA Runtime MUST transition the component implementation to the	Deleted: When the injection [10]
	Destroying state.	Field Code Changed
[JCA40011]	When the component implementation enters the Initializing State, the	Deleted: If an exception is th [11]
	SCA Runtime MUST call the method annotated with @Init on the component implementation, if present.	Field Code Changed
		Deleted: When the compone [12]
[JCA40012]	The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component implementation instance is in the	Field Code Changed
	Initializing state.	Deleted: If a component [13]
[JCA40013]	Once the method annotated with @Init completes successfully, the SCA	Field Code Changed
	Runtime MUST transition the component implementation to the Running state.	Deleted: If a component [14]
[10]	V	Field Code Changed
[JCA40014]	If an exception is thrown whilst initializing, the SCA Runtime MUST transition the component implementation to the Running state.	Deleted: The SCA Runtime I [15]
[10440045]		Field Code Changed
[JCA40015]	If an exception is thrown whilst initializing, the SCA Runtime MUST transition the component implementation to the Destroying state.	Deleted: Once the method [16]
		Field Code Changed
[JCA40016]	When the component implementation scope ends, the SCA Runtime MUST transition the component implementation to the Destroying state.	Deleted: The SCA Runtime [[17]
F I O A 400471		Field Code Changed
[JCA40017]	When a component implementation scope ends, the SCA Runtime MUST transition the component implementation, if present.	Deleted: The SCA Runtime ([18]
		Field Code Changed
[JCA40018]	When a component implementation enters the Destroying state, the SCA Runtime MUST call the method annotated with @Destroy on the	Deleted: theenters the De [19]
	component implementation, if present.	Field Code Changed
[[[] 40040]	The SCA Runtime MUST NOT invoke Service methods on the component	Deleted: If a component [20]
[JCA40019]	implementation when the component implementation instance is in the	Field Code Changed
	Destroying state.	Deleted: If a component [21]
[JCA40020]	Once the method annotated with @Destroy completes successfully, the SCA	Field Code Changed
	Runtime MUST transition the component implementation to the Terminated state.	Deleted: The SCA Runtime I [22]
T104 400047		Field Code Changed
[JCA40021]	If an exception is thrown whilst destroying, the SCA Runtime MUST transition the component implementation to the Terminated state.	Deleted: Once the method [23]
		Field Code Changed
[JCA40022]	The SCA Runtime MUST transition the component implementation to	Deleted: If an exception is th [24]

	_			
tha	10	rmın	2tod	state.
uic	10		aıcu	State.

[JCA40023]

[JCA60003]

JCA600061

*	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.

The interface is annotated with the "asynclnyocation" intent.

The SCA asynchronous service Java interface mapping of a WSDL

request-response operation MUST appear as follows:

The SCA Runtime MUST NOT invoke Service methods on the

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 yped 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.

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

[JCA60005]

If the SCA asynchronous service interface ResponseDispatch handleResponse method is invoked more than once through either its sendResponse or its sendFault method, the SCA runtime MUST throw an IllegalStateException.

follows

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

Asynchronous service methods are characterized by:

- void return type
- a method name with the suffix "Async"

Field Code Changed

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

Field Code Changed

Deleted: 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

Field Code Changed

Deleted: 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.

Field Code Changed

Deleted: 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 t ... [25]

Field Code Changed

Deleted: The @PolicySets annotation MUST NOT be applied to the following:¶ [... [26]

Field Code Changed

Deleted: If the @PolicySets annotation is specified on both an interface method and the meth ... [27]

Field Code Changed

Deleted: The ComponentContext.getService method MUST throw an

Field Code Changed

Deleted: The

ComponentContext.getRequestContext method MUST return non-n ... [29]

[28]

	 a last input parameter with a type of ResponseDispatch<x></x> 		Field Cade Chammed
	 annotation with the asynchrocation intent 	,	Field Code Changed
	 possible annotation with the @AsyncFault annotation 	- 'j	Deleted: When invoked during the execution of a service operation, the getServiceReference method MUST
	The mapping of each such method is as if the method had the return	17	return a ServiceReference that
	type "X", the method name without the suffix "Async" and all the input	11	represents the service that was invoked. When invoked during the
	<u>parameters except the last parameter of the type</u> ResponseDispatch <x>, plus the list of exceptions contained in the</x>	11	execution of a callback operation, the
	@AsyncFault annotation.	,	getServiceReference method MUST return a ServiceReference that
[JCA70001]	SCA identifies annotations that correspond to intents by providing an		represents the callback that was invoked.
	@Intent annotation which MUST be used in the definition of a specific		Field Code Changed
	intent annotation.	/	Deleted: Intent annotations MUST
[JCA70002]	 An SCA runtime MUST verify the proper use of all SCA annotations and if 	//	NOT be applied to the following:
<u></u>	an annotation is improperly used, the SCA runtime MUST NOT run the	/	<#>A method of a service
	component which uses the invalid implementation code.		implementation class, except for a setter method that is either annotated
[JCA70003]	SCA annotations (general or specific) are applied to the same Java		with @Reference or introspec [30]
*	element, the SCA runtime MUST NOT instantiate such an implementation		Field Code Changed
	<mark>class.</mark> ,		Deleted: Where multiple intent
[JCA70004]	If a constructor of an implementation class is annotated with @Constructor and	11, 1	Deleted: MUST NOT be used on
100/1100011	the constructor has parameters, each of these parameters MUST have either a	1, 11	static methods
	@Property annotation or a @Reference annotation.	$\langle \cdot \cdot \rangle$	Deleted: on static fields. It is an error
[JCA70005]	A method annotated with @Destroy MAY have any access modifier and	' ' '	Deleted: use an SCA annotation on
233141631	MUST have a void return type and no arguments.	111	a static method or a static field [[31]
[][0.470006]	If there is a method annotated with @Destroy that matches the criteria for the	1/1	Deleted: compute the combined
[JCA70006]	annotation, the SCA runtime MUST call the annotated method when the scope	1 1	intents for the Java element b [[32]
	defined for the implementation class ends.	11 1	Field Code Changed
[JCA80001]		111	Deleted: If intent annotations are
	<u> </u>	1 1 1 1	specified on both an interface [33]
[JCA80002]	When marked for eager initialization with an @EagerInit annotation, the composite scoped instance MUST be created when its containing component	1 1 1	Field Code Changed
	is started.	$i^{\prime\prime}i^{\prime\prime}$	Deleted: The @PolicySets
1.0.4.000.001		1, 1,	annotation MUST NOT be apr [34]
[JCA80003]	When invoked during the execution of a service operation, the RequestContext.getServiceReference method MUST return a	111	Field Code Changed
	ServiceReference that represents the service that was invoked.	(\cdot,\cdot)	Deleted: If the @PolicySets
		111	annotation is specified on bott [35]
[JCA80004]	The ComponentContext.getServiceReference method MUST throw an	1 1 1	Deleted: The ComponentContext.getServic [36]
	IllegalArgumentException if the reference named by the	11 11	Field Code Changed
	referenceName parameter has multiplicity greater than one.	111	
[JCA80005]	The ComponentContext.getServiceReference method MUST throw an	$\frac{1}{l}$	Deleted: The ComponentContext.getReque [37]
	IllegalArgumentException if the reference named by the	$\frac{I_1}{I_1} = \frac{I_1}{I_1}$	Field Code Changed
	referenceName parameter does not have an interface of the type	11 1	
	defined by the businessInterface parameter.	I_I	Deleted: A method marked with the @Init annotation MAY have an [38]
[JCA80006]	The ComponentContext.getServiceReference method MUST throw an	'\ 	Field Code Changed
	IllegalArgumentException if the component does not have a reference	, 1	Deleted: If there is a method
	with the name provided in the referenceName parameter.		annotated with @Init that mat([39]
[JCA80007][JCA80007]	The ComponentContext.getServiceReference method MUST return	1	Field Code Changed
	null if the multiplicity of the reference named by the referenceName	N.	Deleted: The @Property annotation
	parameter is 01 and the reference has no target service configured.		MUST NOT be used on a clas [40]
[JCA80008]	The ComponentContext.getURI method MUST return the absolute URI		Field Code Changed
<u> </u>	of the component in the SCA Domain.		Deleted: For a @Property
			annotation applied to a constr [41]

[JCA80009]	The ComponentContext.getService method MUST return the proxy		Field Code Changed
	object implementing the interface provided by the businessInterface		Deleted: For a @Property
	parameter, for the reference named by the referenceName parameter		annotation applied to a constructor parameter, the required attribute
	with the interface defined by the businessInterface parameter when		MUST have the value true.
	that reference has a target service configured.		
JCA80010]	The ComponentContext.getService method MUST return null if the		Field Code Changed
	multiplicity of the reference named by the referenceName parameter is		Deleted: The @Qualifier annotation MUST be used in a specific intent
	01 and the reference has no target service configured.		annotation definition where the intent
[JCA80011]	The ComponentContext.getService method MUST throw an		has qualifiers.
	IllegalArgumentException if the component does not have a reference		Field Code Changed
	with the name supplied in the referenceName parameter.	,	Deleted: The @Reference annotation MUST NOT be used on a
JCA80012]	The ComponentContext.getService method MUST throw an		class field that is declared as final.
	IllegalArgumentException if the service reference with the name		Field Code Changed
	supplied in the referenceName does not have an interface compatible	/	Deleted: For a @Reference
	with the interface supplied in the businessInterface parameter.	11	annotation applied to a constructor
[JCA80013]	The ComponentContext.getServiceReference method MUST return a		parameter, there is no default for the
	ServiceReference object typed by the interface provided by the	_	name attribute and the name attribute MUST be present.
	businessInterface parameter, for the reference named by the		Field Code Changed
	referenceName parameter with the interface defined by the	/	
	businessInterface parameter when that reference has a target service configured.	11	Deleted: For a @Reference annotation applied to a constructor
		1/	parameter, the required attribute
[JCA80014]	The ComponentContext.getServices method MUST return a collection		MUST have the value true.
	containing one proxy object implementing the interface provided by the	,	Field Code Changed
	businessInterface parameter for each of the target services configured on the reference identified by the referenceName parameter.	/,	Deleted: If the type of a reference is
	on the reference identified by the reference variety parameter.	11	not an array or any type that extends or implements java.util.Collection,
[JCA80015]	The ComponentContext.getServices method MUST return an empty		then the SCA runtime MUST
	collection if the service reference with the name supplied in the		introspect the component type of the implementation with a <reference></reference>
	referenceName parameter is not wired to any target services.		element with @multiplicity= 01 if the
[JCA80016]	The ComponentContext.getServices method MUST throw an		@Reference annotation required attribute is false and with
	IllegalArgumentException if the reference identified by the	- //	@multiplicity=11 if the @Reference
	referenceName parameter has multiplicity of 01 or 11.		annotation required attribute is true.
[JCA80017]	The ComponentContext.getServices method MUST throw an		Field Code Changed
	IllegalArgumentException if the component does not have a reference		Deleted: If the type of a reference is
	with the name supplied in the referenceName parameter.	- ¬ '\\	defined as an array or as any type that extends or implements [42]
The	The ComponentContext.getServices method MUST throw an	11/1/	Field Code Changed
ComponentContext.getServices	IllegalArgumentException if the service reference with the name	-7///	Deleted: An unwired
method MUST throw an	supplied in the referenceName does not have an interface compatible	1, 1,	
IllegalArgumentException if the	with the interface supplied in the businessInterface parameter.	1111	Deleted: a multiplicity of 01 MUST be presented to the implemen [43]
service reference with the		1,11	- · · · · · · · · · · · · · · · · · · ·
name supplied in the referenceName does not have		1,11	via injection or via API call).
an interface compatible with the		1,11	Field Code Changed
interface supplied in the		1,1	Deleted: An unwired
businessInterface parameter.[JCA80018]		\ \ \	Deleted: a multiplicity of 0n MUST
	The Occurrence (Occupant on Occupant D. Communication D.	\	be presented to the implemen [44
[JCA80019]	<u>The ComponentContext.qetServiceReferences method MUST return a</u> collection containing one ServiceReference object typed by the	-	Deleted: SCA runtime as an empty array or empty collection (eith [45]
	interface provided by the businessInterface parameter for each of the		Field Code Changed
	target services configured on the reference identified by the	Ň	Deleted: References MAY be
			reinjected by an SCA runtime [46]
sca-javacaa-1 1-spec-cd04	06 Feb 201	0	

	referenceName parameter.	,	Field Code Changed
[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.		Deleted: In order for reinjection to occur, the following MUST be true: 1. The component MUST NOT be
[JCA80021]	The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the reference identified by the referenceName parameter has multiplicity of 01 or 11.	, , ,	STATELESS scoped.¶ 2. The reference MUST use either field-based injection or setter injection. References that are injected through constructor injection MUST
[JCA80022]	The ComponentContext.getServiceReferences method MUST throw an		NOT be changed. Field Code Changed
	IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter.	١,	Deleted: If a reference target
[JCA80023]			changes and the reference is not reinjected, the reference MUST continue to work as if the reference target was not changed.
	with the interface supplied in the businessInterface parameter.	, ,	Field Code Changed
[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		Deleted: If an operation is called on a reference where the target of that reference has been undeployed, the SCA runtime SHOULD throw an InvalidServiceException.
	parameter.	j 1	Field Code Changed
[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.		Deleted: 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.
[JCA80026]	The ComponentContext.createSelfReference method MUST return a	1 1	Field Code Changed
	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.		Deleted: If the target service of the reference is changed, the reference MUST either continue to work or throw an InvalidServiceException when it is invoked.
[JCA80027]	The ComponentContext.createSelfReference method MUST throw an	1/ 1/	Field Code Changed
	IllegalArgumentException if the component does not have a service with the name identified by the serviceName parameter.	, , , , , , , , , , , , , , , , , , ,	Deleted: A ServiceReference that has been obtained from a refe [47]
[JCA80028]	The ComponentContext.createSelfReference method MUST throw an	$\frac{1}{l_1} - \frac{1}{l_2}$	Field Code Changed
	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		Deleted: If the target of a ServiceReference has been [48] Field Code Changed
[JCA80029]	businessInterface parameter. The ComponentContext.getProperty method MUST return an object of the type identified by the type parameter containing the value specified \(\).		Deleted: If the target of a ServiceReference has becom [49]
	in the component configuration for the property named by the propertyName parameter or null if no value is specified in the configuration.		Field Code Changed Deleted: If the target service of a ServiceReference is changed [50]
[JCA80030]	The ComponentContext.getProperty method MUST throw an IllegalArgumentException if the component does not have a property	\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \	Field Code Changed Deleted: A reference or ServiceReference accessed tf [51]
	with the name identified by the propertyName parameter.		Field Code Changed
[ICA80031]	The ComponentContext.getProperty method MUST throw an		
[JCA80031]	IllegalArgumentException if the component property with the name identified by the propertyName parameter does not have a type which		Deleted: If the target of a reference or ServiceReference accesse [52] Field Code Changed
	is compatible with the supplied type parameter.	``	Deleted: If the target service of a reference or ServiceReference [53]

[JCA80032]	The ComponentContext.cast method MUST return a ServiceReference		Field Code Changed
A	object which is typed by the same business interface as specified by		Deleted: in the cases where
	the reference proxy object supplied in the target parameter.		reference reinjection is not allowed, the array or Collection for a reference
[JCA80033]	The ComponentContext.cast method MUST throw an		of multiplicity 0n or multiplicity 1n
<u>100A000301</u>	IllegalArgumentException if the supplied target parameter is not an		MUST NOT change its contents when changes occur to the reference wiring
	SCA reference proxy object.		or to the targets of the wiring.
[JCA80034]	The RequestContext.getSecuritySubject method MUST return the		Field Code Changed
<u> </u>	JAAS subject of the current request, or null if there is no subject or null		Deleted: In cases where the
	if the method is invoked from code not processing a service request or	λ	contents of a reference array or collection change when the wiring
	callback request.	, }	changes or the targets change, then
[JCA80035]	The RequestContext.getServiceName method MUST return the name	I_I	for references that use setter injection, the setter method MUST be
	of the service for which an operation is being processed, or null if	I_I	called by the SCA runtime for any
	invoked from a thread that is not processing a service operation or a	" 1,	change to the contents.
	callback operation.	1) ,	Field Code Changed
[JCA80036]	The RequestContext.getCallbackReference method MUST return a	11	Deleted: A reinjected array or Collection for a reference MUST NOT
	ServiceReference object typed by the interface of the callback supplied	11	be the same array or Collection object
	by the client of the invoked service, or null if either the invoked service	11	previously injected to the component.
	is not bidirectional or if the method is invoked from a thread that is not processing a service operation.	1 1	Field Code Changed
		1/	Deleted: A remotable service can be
[JCA80037]	The RequestContext.getCallback method MUST return a reference	1/	published externally as a service and MUST be translatable into a WSDL
	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	1/	portType.
	bidirectional or if the method is invoked from a thread that is not	1 1	Field Code Changed
	processing a service operation.	1/	Deleted: The @Scope annotation
[10.000.00]	When involved during the averaging of a collegely expection the	I_I	MUST only be used on a service's implementation class. It is an error to
[JCA80038]	When invoked during the execution of a callback operation, the RequestContext.getServiceReference method MUST return a	$-\frac{I_I}{I_I}$	use this annotation on an interface.
	ServiceReference that represents the callback that was invoked.	1	Field Code Changed
[JCA80039]	When invoked from a thread not involved in the execution of either a	1) 1)	Deleted: An implementation class
<u> 304000391</u>	service operation or of a callback operation, the	1/	need not be declared as
	RequestContext.getServiceReference method MUST return null.	11	implementing all of the interfaces implied by the services declared in its
[[] [] [] [] []	The ServiceReference.getService method MUST return a reference	11 11	@Service annotation, but all r [54]
[JCA80040]	proxy object which can be used to invoke operations on the target	1/ 1	Field Code Changed
	service of the reference and which is typed with the business interface		Deleted: A @Service annota [55]
	of the reference.	11 1	Field Code Changed
[JCA80041]	The ServiceReference.getBusinessInterface method MUST return a	1,	Deleted: A @Service annota [56]
	Class object representing the business interface of the reference.	, , ,	Field Code Changed
[JCA80042]	The SCAClientFactory.newInstance(URI) method MUST return an		Deleted: A component [57]
	object which implements the SCAClientFactory class for the SCA	. \	Field Code Changed
	Domain identified by the domainURI parameter.	1, ,	Deleted: When used to anno [58]
[JCA80043]	The SCAClientFactory.newInstance(URI) method MUST throw a	1	Field Code Changed
	NoSuchDomainException if the domainURI parameter does not identify	· · ·	Deleted: For a @Property [59]
	a valid SCA Domain.		Field Code Changed
[JCA80044]	The SCAClientFactory.newInstance(Properties, URI) method MUST	×	Deleted: If the name attribute [60]
	return an object which implements the SCAClientFactory class for the	Ţ.,	Field Code Changed
	SCA Domain identified by the domainURI parameter.	1	Deleted: If the names attribu [61]
[<u>JCA80045]</u>	The SCAClientFactory.newInstance(Properties, URI) method MUST		Field Code Changed
	throw a NoSuchDomainException if the domainURI parameter does	``\.	
			Deleted: The number of Strir [62]

	not identify a valid SCA Domain.		, { <u>F</u>	Field Code Changed
[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.	۔ '۔'۔'	N C	Deleted: The @Service annotation MUST NOT specify Void.class in conjunction with any other service class or interface.
[JCA80047]	The SCAClientFactory.newInstance(Classloader, URI) method MUST throw a NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.	-<[[C	Field Code Changed Deleted: The @AllowsPassByReference annotation MAY be placed on an individual
[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.	, , , , , , , , , , , , , , , , , , ,	ir ir ir s	nethod of a remotable service mplementation, on a service mplementation class, or on an individual reference for a remotable service. When applied to a reference, t MAY appear anywhere that the
[JCA80049]	The SCAClientFactory.newInstance(Properties, Classloader, URI) MUST throw a NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.	- 'i'	a \\a \\\a	Remotable annotation MAY appear It MUST NOT appear anywhere else.
[JCA80050]	The SCAClientFactory.getService method MUST return a proxy object	1	$^{\prime}$	Field Code Changed
JJCA80051]	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. The SCAClientFactory.getService method MUST throw a		J th ir ir	Deleted: For the purposes of the lava-to-WSDL mapping algorithm, he SCA runtime MUST treat a Java nterface as if it had a @WebService annotation on the class, even if it doesn't.
<u> </u>	NoSuchServiceException if a service with the relative URI serviceURI and a business interface which matches interface cannot be found in the SCA Domain targeted by the SCAClient object.			Field Code Changed Deleted: The SCA runtime MUST reat an
[JCA80052]	The SCAClientFactory.getService method MUST throw a NoSuchServiceException if the domainURI of the SCAClientFactory does not identify a valid SCA Domain.	- 1 - 1 - 1 - 1	\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\	@org.oasisopen.sca.annotation.One Way annotation as a synonym for the @javax.jws.OneWay annotation. Field Code Changed
[JCA80053]	The SCAClientFactory.getDomainURI method MUST return the SCA Domain URI of the Domain associated with the SCAClientFactory object.	1	n ta	Deleted: For the WSDL-to-Java mapping, the SCA runtime MUST ake the generated @WebService annotation to imply that the Java
[JCA80054]	The SCAClientFactory.getDomainURI method MUST throw a	- 1 1 1 1 1 1	11/	nterface is @Remotable.
The implementation of the	NoSuchServiceException if the domainURI of the SCAClientFactory does not identify a valid SCA Domain. The implementation of the SCAClientFactoryFinder.find method MUST	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	L S	Deleted: SCA runtimes MUST support the JAXB 2.1 mapping from lava types to XML schema types.
SCAClientFactoryFinder.find	return an object which is an implementation of the SCAClientFactory	- j - j - - j -	F	ield Code Changed
method MUST return an object which is an implementation of the SCAClientFactory interface,	interface, for the SCA Domain represented by the doaminURI parameter, using the supplied properties and classloader.		i th	Deleted: SCA runtimes MAY support he SDO 2.1 mapping from Java ypes to XML schema types.
for the SCA Domain represented by the doaminURI		1 1	F	ield Code Changed
parameter, using the supplied properties and classloader. [JCA80055]		11 11 11	d ir	Deleted: For SCA service interfaces defined using interface.java, the Java nterface MUST NOT contain [63]
[JCA80056]	The implementation of the SCAClientFactoryFinder.find method MUST throw a ServiceRuntimeException if the SCAClientFactory	-\ <u>\</u>	֓֞֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓֓	Field Code Changed Deleted: For SCA reference nterfaces defined using [64]
	implementation could not be found.	1/ 1/	F	ield Code Changed
[JCA50057]	The ResponseDispatch.sendResponse() method MUST send the response message to the client of an asynchronous service.	1,	\\ <mark>a</mark>	Deleted: If the additional client-side asynchronous polling and call [65]
[JCA80058]	The ResponseDispatch.sendResponse() method MUST throw an InvalidStateException if either the sendResponse method or the		\ 	Deleted: SCA runtimes MUST support the use of the JAX-W[[66]
				support the use of the JAX-W([66]

	sendFault method has already been called once.
[JCA80059]	The ResponseDispatch.sendFault() method MUST send the supplied fault to the client of an asynchronous service.
[JCA80060]	The ResponseDispatch.sendFault() method MUST throw an InvalidStateException if either the sendResponse method or the sendFault method has already been called once.
[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.
[JCA90001]	SCA annotations MUST NOT be used on static methods or on static fields. It is an error to use an SCA annotation on a static method or a static field of an implementation class and the SCA runtime MUST NOT instantiate such an implementation class.
[JCA90003]	If a constructor of an implementation class is annotated with @Constructor and the constructor has parameters, each of these parameters MUST have either a @Property annotation or a @Reference annotation.
[JCA90004]	A method annotated with @Destroy can have any access modifier and MUST have a void return type and no arguments.
[JCA90005]	If there is a method annotated with @Destroy that matches the criteria for the annotation, the SCA runtime MUST call the annotated method when the scope defined for the implementation class ends.
[JCA90007]	When marked for eager initialization with an @EagerInit annotation, the composite scoped instance MUST be created when its containing component is started.
[JCA90008]	A method marked with the @Init annotation can have any access modifier and MUST have a void return type and no arguments.
[JCA90009]	If there is a method annotated with @Init that matches the criteria for the annotation, the SCA runtime MUST call the annotated method after all property and reference injection is complete.
[JCA90011]	The @Property annotation MUST NOT be used on a class field that is declared as final.
[JCA90013]	For a @Property annotation applied to a constructor parameter, there is no default value for the name attribute and the name attribute MUST be present.
[JCA90014]	For a @Property annotation applied to a constructor parameter, the required attribute MUST NOT have the value false.
[JCA90015]	The @Qualifier annotation MUST be used in a specific intent annotation definition where the intent has qualifiers.
[JCA90016]	The @Reference annotation MUST NOT be used on a class field that is declared as final.
[JCA90018]	For a @Reference annotation applied to a constructor parameter, there is no default for the name attribute and the name attribute MUST be present.
[JCA90019]	For a @Reference annotation applied to a constructor parameter, the

	required attribute MUST have the value true.
[JCA90020]	If the type of a reference is not an array or any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference></reference> element with @multiplicity= 01 if the @Reference annotation required attribute is false and with @multiplicity=11 if the @Reference annotation required attribute is true.
[JCA90021]	If the type of a reference is defined as an array or as any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference></reference> element with @multiplicity=0n if the @Reference annotation required attribute is false and with @multiplicity=1n if the @Reference annotation required attribute is true.
[JCA90022]	An unwired reference with a multiplicity of 01 MUST be presented to the implementation code by the SCA runtime as null (either via injection or via API call).
[JCA90023]	An unwired reference with a multiplicity of 0n MUST be presented to the implementation code by the SCA runtime as an empty array or empty collection (either via injection or via API call).
[JCA90024]	References MAY be reinjected by an SCA runtime after the initial creation of a component if the reference target changes due to a change in wiring that has occurred since the component was initialized.
[JCA90025]	 In order for reinjection to occur, the following MUST be true: The component MUST NOT be STATELESS scoped. The reference MUST use either field-based injection or setter injection. References that are injected through constructor injection MUST NOT be changed.
[JCA90026]	If a reference target changes and the reference is not reinjected, the reference MUST continue to work as if the reference target was not changed.
[JCA90027]	If an operation is called on a reference where the target of that reference has been undeployed, the SCA runtime SHOULD throw an InvalidServiceException.
[JCA90028]	If an operation is called on a reference where the target of the reference has become unavailable for some reason, the SCA runtime SHOULD throw a ServiceUnavailableException.
[JCA90029]	If the target service of the reference is changed, the reference MUST either continue to work or throw an InvalidServiceException when it is invoked.
[JCA90030]	A ServiceReference that has been obtained from a reference by ComponentContext.cast() corresponds to the reference that is passed as a parameter to cast(). If the reference is subsequently reinjected, the ServiceReference obtained from the original reference MUST continue to work as if the reference target was not changed.
[JCA90031]	If the target of a ServiceReference has been undeployed, the SCA runtime SHOULD throw a InvalidServiceException when an operation is invoked on the ServiceReference.

[JCA90032]	If the target of a ServiceReference has become unavailable, the SCA runtime SHOULD throw a ServiceUnavailableException when an operation is invoked on the ServiceReference.
[JCA90033]	If the target service of a ServiceReference is changed, the reference MUST either continue to work or throw an InvalidServiceException when it is invoked.
[JCA90034]	A reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() MUST correspond to the current configuration of the domain. This applies whether or not reinjection has taken place.
[JCA90035]	If the target of a reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() has been undeployed or has become unavailable, the result SHOULD be a reference to the undeployed or unavailable service, and attempts to call business methods SHOULD throw an InvalidServiceException or a ServiceUnavailableException.
[JCA90036]	If the target service of a reference or ServiceReference accessed through the component context by calling getService() or getServiceReference() has changed, the returned value SHOULD be a reference to the changed service.
[JCA90037]	in the cases where reference reinjection is not allowed, the array or Collection for a reference of multiplicity 0n or multiplicity 1n 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
sca-javacaa-1 1-spec-cd04	06 Feb 2010

	java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <pre><pre>component type of the implementation with a <pre>component</pre></pre></pre>
[JCA90050]	a @many attribute set to true, otherwise @many MUST be set to false. The number of Strings in the names attribute array of the @Service
	annotation MUST match the number of elements in the value attribute array.
[JCA90052]	The @AllowsPassByReference annotation MUST only annotate the following locations:
	 a service implementation class an individual method of a remotable service implementation
	an individual reference which uses a remotable interface, where the reference is a field, a setter method, or a constructor parameter.
[JCA90053]	The @Remotable annotation is valid only on a Java interface, a Java class, a field, a setter method, or a constructor parameter. It MUST NOT appear anywhere else.
[JCA90054]	When used to annotate a method or a field of an implementation class for injection of a callback object, the type of the method or field MUST be the callback interface 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 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.

4153

D. Acknowledgements

4155 The following individuals have participated in the creation of this specification and are gratefully 4156

acknowledged:

4154

4157

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

Martin Chapman

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**

SAP AG* Rainer Kerth 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.

SAP AG* Vladimir Savchenko

Pradeep Simha TIBCO Software Inc. Raghav Srinivasan **Oracle Corporation**

Scott Vorthmann Feng Wang Robin Yang

4158

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

Deleted: ¶

<#>Non-Normative Text¶

Formatted: Bullets and Numbering

4160

E. Revision History

[optional; should not be included in OASIS Standards]

4161 4162

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.

sca-javacaa-1.1-spec-cd04
Copyright © OASIS® 2005, 2010. All Rights Reserved.

06 Feb 2010 Page 124 of 126

			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
JULIEVU	2000 04-20	WING Edwards	Issue 98 - Section 8
cd02-rev7	2009-04-30	Mike Edwards	Editorial cleanup throughout the spec
5302 10V1	2000 07 00	mino Edwards	Landinal didaliap undagilout tilo spoo

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
ad00 ray0a	2000 05 02	Simon Nash	Minor editorial alconus
cd02-rev8a	2009-05-03	Simon wash	Minor editorial cleanup
cd03	2009-05-04	Anish Karmarkar	Updated references and front page clean up
cd03-rev1	2009-09-15	<u>David Booz</u>	Applied Issues: 1,13,125,131,156,157,158,159,161,165,172,177
cd03-rev2	2010-01-19	<u>David Booz</u>	<u>Updated to current Assembly namespace</u> <u>Applied issues:</u> 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
<u>CD04</u>	2010-02-06	Dave Booz	Editorial updates for Committee Draft 04 All changes accepted

Page 107: [1] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
perform any constructor reference or prope	erty injection when it calls the	
Page 107: [2] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
at the start of the Constructing state		
Page 107: [3] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
The SCA Runtime MUST perform any constructor of a component implementation.		y injection when it calls the
constructor of a component implement	<u> </u>	
Page 107: [4] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
If an exception is thrown whilst in		
Page 107: [5] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
constructor completes successfully		_,
Page 107: [6] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
When a component implementation instance	<mark>ce</mark>	
Page 107: [7] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
first inject all field and setter properties that	are present into	
Page 108: [8] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
first		
Page 108: [8] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
references		
Page 108: [8] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
, after all the properties have been injected		
Page 108: [9] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
ensure that the correct synchronization	n model is used so that all injecte	ed properties and
references are made visible to		
Page 108: [9] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
without requiring		
Page 108: [9] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
developer to do any specific synchroniz	zation.	
Page 108: [10] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
When the injection of properties and referen	nces completes successfully, the	
Page 108: [10] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
NOT invoke Service methods		
Page 108: [10] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
when the component implementation is		
		

Page 108: [11] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
If an exception is thrown whilst injecting	2010. 00	_,,
in an exception to the will write injecting		
Page 108: [11] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
and		
Page 108: [11] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
completes successfully		
Page 108: [11] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
<u>Initializing</u>		
Page 108: [12] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
When the component implementation enters t		ime MUST call the method
annotated with @Init on the component imple	mentation, if present.	
Page 108: [13] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
If a component implementation invokes an op-		
not yet been initialized, the SCA Runtime MUS		
not you soon minaneou, and contraining most	57 III 611 G 5617165671G G 11G 156276	option.
Page 108: [14] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
If a component implementation invokes a	n operation on an injected refe	rence that refers to a
target that has not yet been initialized, the		
ServiceUnavailableException.		
Page 108: [15] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
The SCA Runtime MUST NOT invoke Se		ent implementation when
the component implementation instance i	s in the Initializing state.	
Page 108: [16] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
Once the method annotated with @Init co		
	ompletes successfully	
Provided Edit Political		0/4//0040 40 40 00 8M
Page 108: [16] Deleted	Mike Edwards	2/16/2010 10:10:00 AM
Page 108: [16] Deleted Destroying		2/16/2010 10:10:00 AM
Destroying	Mike Edwards	
Destroying Page 108: [17] Deleted	Mike Edwards Mike Edwards	2/16/2010 10:10:00 AM
Page 108: [17] Deleted The SCA Runtime MUST invoke Service meth	Mike Edwards Mike Edwards ods on a component implementat	2/16/2010 10:10:00 AM ion instance when the
Destroying Page 108: [17] Deleted	Mike Edwards Mike Edwards ods on a component implementat	2/16/2010 10:10:00 AM ion instance when the
Page 108: [17] Deleted The SCA Runtime MUST invoke Service method component implementation is in the Running state.	Mike Edwards Mike Edwards ods on a component implementat	2/16/2010 10:10:00 AM ion instance when the
Page 108: [17] Deleted The SCA Runtime MUST invoke Service method component implementation is in the Running state.	Mike Edwards Mike Edwards ods on a component implementat	2/16/2010 10:10:00 AM ion instance when the
Page 108: [17] Deleted The SCA Runtime MUST invoke Service method component implementation is in the Running sthe component. Page 108: [18] Deleted The SCA Runtime MUST invoke Service	Mike Edwards Mike Edwards ods on a component implementate state and a client invokes operation Mike Edwards methods on a component implementation	2/16/2010 10:10:00 AM ion instance when the ns on a service offered by 2/16/2010 10:10:00 AM ementation instance
Page 108: [17] Deleted The SCA Runtime MUST invoke Service method component implementation is in the Running sthe component. Page 108: [18] Deleted The SCA Runtime MUST invoke Service when the component implementation is in	Mike Edwards Mike Edwards ods on a component implementate state and a client invokes operation Mike Edwards methods on a component implementation	2/16/2010 10:10:00 AM ion instance when the ns on a service offered by 2/16/2010 10:10:00 AM ementation instance
Page 108: [17] Deleted The SCA Runtime MUST invoke Service method component implementation is in the Running sthe component. Page 108: [18] Deleted The SCA Runtime MUST invoke Service	Mike Edwards Mike Edwards ods on a component implementate state and a client invokes operation Mike Edwards methods on a component implementation	2/16/2010 10:10:00 AM ion instance when the ns on a service offered by 2/16/2010 10:10:00 AM ementation instance
Page 108: [17] Deleted The SCA Runtime MUST invoke Service metromponent implementation is in the Running sthe component. Page 108: [18] Deleted The SCA Runtime MUST invoke Service when the component implementation is in service offered by the component.	Mike Edwards Mike Edwards nods on a component implementate state and a client invokes operation Mike Edwards methods on a component implementation of the Running state and a client	2/16/2010 10:10:00 AM ion instance when the ns on a service offered by 2/16/2010 10:10:00 AM ementation instance t invokes operations on a
Page 108: [17] Deleted The SCA Runtime MUST invoke Service method component implementation is in the Running sthe component. Page 108: [18] Deleted The SCA Runtime MUST invoke Service when the component implementation is in service offered by the component. Page 108: [19] Deleted	Mike Edwards Mike Edwards ods on a component implementate state and a client invokes operation Mike Edwards methods on a component implementation	2/16/2010 10:10:00 AM ion instance when the ns on a service offered by 2/16/2010 10:10:00 AM ementation instance
Page 108: [17] Deleted The SCA Runtime MUST invoke Service metromponent implementation is in the Running sthe component. Page 108: [18] Deleted The SCA Runtime MUST invoke Service when the component implementation is in service offered by the component.	Mike Edwards Mike Edwards nods on a component implementate state and a client invokes operation Mike Edwards methods on a component implementation of the Running state and a client	2/16/2010 10:10:00 AM ion instance when the ns on a service offered by 2/16/2010 10:10:00 AM ementation instance t invokes operations on a
Page 108: [17] Deleted The SCA Runtime MUST invoke Service metromponent implementation is in the Running sthe component. Page 108: [18] Deleted The SCA Runtime MUST invoke Service when the component implementation is in service offered by the component. Page 108: [19] Deleted the	Mike Edwards Mike Edwards nods on a component implementate state and a client invokes operation Mike Edwards methods on a component implementation of the Running state and a client Mike Edwards	2/16/2010 10:10:00 AM ion instance when the ns on a service offered by 2/16/2010 10:10:00 AM ementation instance t invokes operations on a 2/16/2010 10:10:00 AM
Page 108: [17] Deleted The SCA Runtime MUST invoke Service method component implementation is in the Running sthe component. Page 108: [18] Deleted The SCA Runtime MUST invoke Service when the component implementation is in service offered by the component. Page 108: [19] Deleted	Mike Edwards Mike Edwards nods on a component implementate state and a client invokes operation Mike Edwards methods on a component implementation of the Running state and a client	2/16/2010 10:10:00 AM ion instance when the ns on a service offered by 2/16/2010 10:10:00 AM ementation instance t invokes operations on a

Page 108: [19] Deleted Mike Edwards 2/16/2010 10:10:00 AM call the method annotated with @Destroy Page 108: [19] Deleted Mike Edwards 2/16/2010 10:10:00 AM to the Destroying state. Page 108: [20] Deleted Mike Edwards 2/16/2010 10:10:00 AM 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. Page 108: [21] Deleted 2/16/2010 10:10:00 AM Mike Edwards 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. Page 108: [22] Deleted Mike Edwards 2/16/2010 10:10:00 AM The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component implementation instance is in the Destroying state. Page 108: [23] Deleted Mike Edwards 2/16/2010 10:10:00 AM Once the method annotated with @Destroy completes successfully Page 108: [24] Deleted Mike Edwards 2/16/2010 10:10:00 AM If an exception is thrown whilst destroying, the 2/16/2010 10:10:00 AM Page 108: [24] Deleted Mike Edwards NOT invoke Service methods Page 108: [24] Deleted Mike Edwards 2/16/2010 10:10:00 AM when the component implementation instance is in Page 109: [25] Deleted Mike Edwards 2/16/2010 10:10:00 AM If intent annotations are specified on both an interface method and the method's declaring interface, the

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.

Page 109: [26] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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

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

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

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

Page 109: [27] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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.

Page 109: [28] Deleted Mike Edwards 2/16/2010 10:10:00 AM

The ComponentContext.getService method MUST throw an IllegalArgumentException if the reference identified by the referenceName parameter has multiplicity of 0..n or 1..n.

Page 109: [29] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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.

Page 110: [30] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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

Page 110: [31] Deleted Mike Edwards 2/16/2010 10:10:00 AM

use an SCA annotation on a static method or a static field of an implementation class and

Page 110: [32] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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.

Page 110: [33] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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.

Page 110: [34] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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

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

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

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

Page 110: [35] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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.

Page 110: [36] Deleted Mike Edwards 2/16/2010 10:10:00 AM

The ComponentContext.getService method MUST throw an IllegalArgumentException if the reference identified by the referenceName parameter has multiplicity of 0...n or 1...n.

Page 110: [37] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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.

Page 110: [38] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

A method marked with the @Init annotation MAY have any access modifier and MUST have a void return type and no arguments.

Page 110: [39] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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.

Page 110: [40] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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

Page 110: [41] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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.

Page 111: [42] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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.

Page 111: [43] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

a multiplicity of 0..1 MUST be presented to the implementation code by

Page 111: [44] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

a multiplicity of 0..n MUST be presented to the implementation code by

Page 111: [45] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

SCA runtime as an empty array or empty collection (either via injection or via API call).

Page 111: [46] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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.

Page 112: [47] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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.

Page 112: [48] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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

Page 112: [49] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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

Page 112: [50] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

If the target service of a ServiceReference is changed, the reference MUST either continue to work or throw an InvalidServiceException when it is invoked.

Page 112: [51] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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

Page 112: [52] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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

Page 112: [53] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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

Page 113: [54] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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.

Page 113: [55] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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

Page 113: [56] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

A @Service annotation that specifies a single class object Void.class either explicitly or by default is equivalent to not having the annotation there at all - such a @Service annotation MUST be ignored.

Page 113: [57] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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

Page 113: [58] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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.

Page 113: [59] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

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.

Page 113: [60] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

If the name attribute is specified on the @Service annotation, the value attribute MUST also be specified.

Page 113: [61] Deleted

Mike Edwards

2/16/2010 10:10:00 AM

If the names attribute is specified for an @Service annotation, the interfaces attribute MUST also be specified.

Page 113: [62] Deleted Mike Edwards 2/16/2010 10:10:00 AM

The number of Strings in the names attributes array of the @Service annotation MUST match the number of elements in the interfaces attribute array.

Page 114: [63] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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.

Page 114: [64] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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

Page 114: [65] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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.

Page 114: [66] Deleted Mike Edwards 2/16/2010 10:10:00 AM

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